普通高等教育“十三五”规划教材

C 语言程序设计

（第 2 版）

耿　姝　主　编

逯　柳　张　旭　孙　毅　副主编

電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内 容 简 介

本书从程序设计的实际能力培养出发，由浅入深、深入浅出，将理论与实践有机结合，集知识传播和能力培养为一体。本书内容丰富、注重实践；突出重点、分散难点；例题广泛、结合实际。本书的宗旨在于进一步巩固对基本知识的理解和掌握，提高学生的逻辑分析、抽象思维和程序设计能力，培养学生良好的程序设计风格，进而具备编写中、大型程序的能力。

本书中的程序在按照模块化程序设计思想进行编写的同时，每一个程序都遵循软件工程方法学的编程风格，即采用缩进格式，程序中附有注释，以便于对程序的分析、理解和自学。

本书不仅可以作为高等院校学生 C 语言程序设计教材，而且还可以作为计算机等级考试的参考书和编程爱好者自学 C 语言的自学教材。

图书在版编目（CIP）数据

C 语言程序设计 / 耿姝主编. —2 版. —北京：电子工业出版社, 2016.2
普通高等教育“十三五”规划教材
ISBN 978-7-121-27845-7

I. ①C… II. ①耿… III. ①C 语言－程序设计－高等学校－教材 IV. ①TP312

中国版本图书馆 CIP 数据核字（2015）第 300421 号

策划编辑：袁 玺
责任编辑：郝黎明 特约编辑：张燕虹
印 刷：
装 订：
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编：100036
开 本：787×1092 1/16 印张：17.5 字数：504 千字
版 次：2012 年 8 月第 1 版
2016 年 2 月第 2 版
印 次：2016 年 2 月第 1 次印刷
定 价：38.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

计算机应用能力是21世纪人才不可缺少的基本素质。程序设计是各专业计算机应用能力培养的重要技术基础，C语言是目前国内外广泛使用的一种程序设计语言，是国内外大学讲述程序设计方法的首选语言。

全国计算机等级考试、全国计算机应用技术证书考试和全国各地区组织的大学生计算机等级考试都已将C语言列入了考试范围。学习C语言已成为广大计算机应用人员和青年学生的迫切愿望和要求。

教材是知识传播和能力培养的基础，本书从程序设计的实际能力培养出发，由浅入深、深入浅出，将理论与实践有机结合，集知识传播和能力培养为一体。本书内容丰富、注重实践；突出重点、分散难点；例题广泛、结合实际。

本书共10章，主要介绍了C语言程序设计基础知识、数据的存储与运算、三种结构化程序设计方法、数组、函数、指针、用户自定义数据类型和文件系统等。每章均配有典型习题，突出了实用性，强调理论与实践相结合，培养了学生的编程能力。

本书不仅可以作为高等院校学生C语言程序设计教材，而且还可以作为计算机等级考试的参考书和编程爱好者自学C语言的自学教材。

本书有大量的算法语句、程序语句及计算公式，对于其中的变量，为了方便读者阅读，避免歧义，不再区分正、斜体，而是统一采用正体，特此说明。

本书由耿姝主编，各章节的编写分工如下：第1章、第2章、第3章由逯柳编写；第4章、第5章、第6章由张旭编写；第7章、第8章及附录由耿姝编写；第9章、第10章由孙毅编写。

由于作者水平有限，书中难免存在疏漏与不妥之处，恳请同行与广大读者批评指正。

编　者

目　录

第 1 章　C 语言程序设计概述

1.1　程序与程序设计语言

1.1.1　程序

计算机是可以按照人们事先编写的程序高速、精确地进行数据加工、处理的电子装置。如果我们需要计算机完成什么工作，将要完成工作的步骤用诸条指令描述出来，并把这些指令存放在计算机的存储器中，需要结果时就向计算机发出一条简单的命令，计算机就会自动逐条顺序地执行指令，全部指令执行完就得到了预期的结果。这种可以被连续执行的一条条指令的集合称为计算机的程序。也就是说，程序是计算机的指令序列，是用计算机语言对所要解决的问题中的数据以及处理问题的方法和步骤所做的完整而准确的描述。

1.1.2　计算机语言

人们用自然语言讲述和书写，目的是给别人传播信息。同样，我们使用计算机语言把我们的意图表达给计算机，目的是使用计算机，让计算机给我们完成一定信息的处理。

为了使计算机进行各种工作，就需要有一套用以编写计算机程序的数字、字符和语法规则，由这些数字、字符和语法规则组成计算机的各种指令（或各种语句），就是计算机能接受的语言。

计算机每做一次动作，完成一个步骤，都是按照已经用计算机语言编好的程序来执行的。计算机语言没有自然语言那么丰富多样，而只是有限规则的集合，所以它简单易学。但是，也正因为它是根据机器的特点编制的，所以交流中无法意会和言传，更多地表现了说一不二，表现了规则的严谨。

1．程序语言的发展

计算机语言不断从低级向高级发展，其发展过程可分为三代：机器语言、汇编语言和高级语言。

机器语言是用二进制代码表示的计算机能直接识别和执行的机器指令的集合。它是计算机的设计者通过计算机的硬件结构赋予计算机的操作功能，它与计算机同时诞生，是第一代的计算机语言。

例如，计算 A=12+10 的机器语言程序如下：

```
10110000 00001100       ：把 12 放入累加器 A 中
00101100 00001010       ：10 与累加器 A 的值相加，结果仍放入 A 中
11110100                ：结束，停机
```

使用机器语言的缺点是编程工作量大，难学、难记、难修改，它只适合专业人员使用；而且由于不同的计算机，其指令系统不同，机器语言随机而异，通用性差，是面向机器的语言。

机器语言的优点是程序代码不需要翻译，所占空间少，执行速度快。

汇编语言将机器指令的二进制代码用英文助记符来表示，代替机器语言中的指令和数据。例如用 ADD 表示加、SUB 表示减、JMP 表示程序跳转等，这种指令助记符的集合就是汇编语言。

例如，计算 A=12+10 的汇编语言程序：

```
MOV A，12          ：把 12 放入累加器 A 中
ADD A，10          ：10 与累加器 A 相加，结果存入 A 中
HLT                ：结束，停机
```

汇编语言的优点是克服了机器语言难读等缺点，保持了其编程质量高，占存储空间少、执行速度快的优点。

汇编语言的缺点是依赖于机器，通用性差。

汇编语言源程序必须通过汇编程序翻译成机器语言，计算机才能执行。汇编语言常用于过程控制等编程。

高级语言是一种接近于自然语言和数学公式的程序设计语言。它采用了完全符号化的描述形式，用类似自然语言的形式描述对问题的处理过程，用数学表达式的形式描述对数据的计算过程。高级语言主要是相对于汇编语言而言，它并不是特指某一种具体的语言，而是包括了很多编程语言。

例如，计算 A=12+10 的 BASIC 语言程序如下：

```
A=12+10            ：12 与 10 相加的结果放入 A 中
PRINT A            ：输出 A
END                ：程序结束
```

高级语言的优点是通用性强，编程效率高。它使程序员可以不用与计算机的硬件打交道，可以不必了解机器的指令系统，集中精力解决问题本身而不受机器制约，极大地提高了编程的效率。

高级语言源程序要通过翻译程序翻译成机器语言才能运行，程序的效率不如优化的汇编程序高。

机器语言和汇编语言都是面向机器的语言，属于低级语言；而高级语言又分为面向过程和面向对象两种。前一种程序设计是数据被加工的过程，后一种程序设计的关键是定义类，并由类派生对象。客观世界可以分类，对象是类的实例，对象是数据和方法的封装，对象间通过发送和接收消息发生联系。

面向过程的高级语言只是要求人们向计算机描述问题的求解过程，而不关心计算机的内部结构，它易于被人们理解和接受。典型的面向过程语言有 BASIC、FORTRAN、COBOL、C、Pascal 等。程序设计语言的发展如图 1-1 所示。

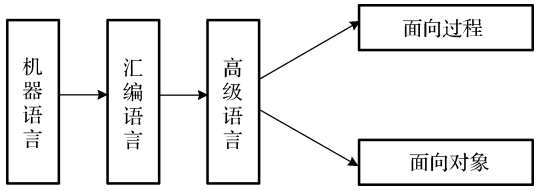


图 1-1　程序设计语言的发展

面向对象的高级语言是“面向过程”的一次革命，如果说面向过程的语言要求人们告诉计算机怎么做，那么面向对象的语言只要求人们告诉计算机做什么。面向对象是通过类和对象把程序所涉及的数据结构和对它施行的操作有机地组织成模块，对数据和数据的处理细节进行最大限度的封装，从而使开发出来的软件易重用、易修改、易调试、易扩充。

2．语言处理程序

在所有的程序设计语言中，除了用机器语言编制的程序能够被计算机直接理解和执行外，其他的程序设计语言编写的源程序都必须经过一个翻译过程才能转换为计算机所能识别的机器语言程序。实现这个翻译过程的工具是语言处理程序，即翻译程序。翻译程序也称为编译器。用非机

器语言写的程序称为源程序，通过翻译程序翻译后的程序称为目标程序。针对不同的程序设计语言编写出的程序，有各自的翻译程序，互相不通用。

翻译程序翻译源程序通常有两种方式：解释方式和编译方式。

（1）解释方式：解释方式的翻译工作由解释程序来完成，这种方式如同“口译”。解释程序对源程序进行逐句分析，若没有错误，将该语句翻译成一个或多个机器语言指令；然后立即执行这些指令；若解释时发现错误，会立即停止，报错并提醒用户更正代码，具体过程如图 1-2 所示，解释方式不生成目标程序。

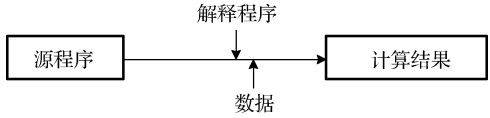


图 1-2　解释方式的执行过程

采用解释方式的优点是查找错误的语句行和修改方便。缺点是执行速度慢，采用解释方式运行的源程序，每次运行都必须重新解释，若程序较大，错误发生在程序的后面，则前面运行的结果是无效的，解释程序无法对整个程序进行优化。

（2）编译方式：翻译工作由编译程序完成。如同“笔译”，在纸上记录翻译后的结果。编译程序过程：对源程序编译产生目标程序，连接程序将目标程序和有关的程序库组合成可执行程序，编译方式的执行过程如图 1-3 所示。

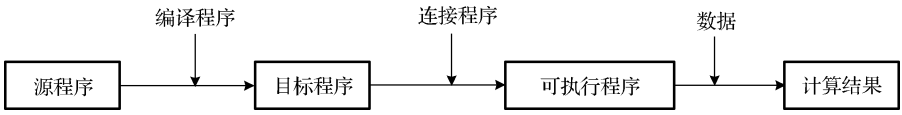


图 1-3　编译方式的执行过程

采用编译方式的优点是程序执行速度快，产生的可执行程序可以脱离编译程序和源程序独立存在并反复运行，但修改源程序后都必须重新编译生成目标程序。

一般高级语言（C/C++、Pascal、FORTRAN、COBOL 等）都是采用编译方式。

1.2　程序设计（解决什么问题、如何解决、实现方法）

计算机之所以能够产生如此大的影响，其原因不仅在于人们发明了机器本身，更重要的是人们为计算机开发出了不计其数的能够指挥计算机完成各种各样工作的程序。正是这些功能丰富的程序给了计算机无尽的生命力，是程序设计工作智慧的结晶。

所谓程序，是用计算机语言对所要解决的问题中的数据以及处理问题的方法和步骤所做的完整而准确的描述，而程序设计就是用某种程序语言编写这些解决问题的步骤的过程。对数据的描述就是指明所要处理问题中的数据结构（即数据和数据之间的关系）；对问题处理方法和步骤的描述就是算法。因此，数据结构与算法是程序设计过程中密切相关的两个方面。著名计算机科学家 Niklaus Wirth 教授关于程序提出了著名公式：程序=数据结构+算法。这个公式说明了程序设计的主要任务。

如何进行程序设计呢？一个简单的程序设计一般包含以下步骤。

1．分析问题

使用计算机解决具体问题时，首先要对问题进行充分的分析，确定问题是什么，解决问题

的步骤又是什么。针对所要解决的问题，找出已知的数据和条件，确定所需的输入、处理及输出对象。

2．确定数据结构和算法

在分析求解问题的基础上，将所研究问题的数据和数据间关系抽象出来，确定程序中数据的类型和数据组织存储形式，即确定存放数据的数据结构。针对问题的分析和确定的数据结构，选择合适的算法加以实现。注意，这里所说的“算法”泛指解决某一问题的方法和步骤。

3．编制程序

根据确定的数据结构和算法，用一种程序设计语言把这个解决方案严格地描述出来，也就是编写出程序代码。

4．调试程序

在计算机上用实际的输入数据对编好的程序进行调试，分析所得到的运行结果，进行程序的测试和调整，直至获得预期的结果。

5．分析结果

对程序执行结果进行验证和分析，发现程序中存在的问题并修改完善。

6．写出程序的文档

程序是提供给用户使用的，如同正式的产品应当提供产品说明书一样，正式提供给用户使用的程序，必须向用户提供程序说明书。内容应包括程序名称、程序功能、运行环境、程序的装入和启动、需要输入的数据，以及使用注意事项等，为程序的使用、修改做好基础工作。

1.3 C 语言的发展

C 语言是世界上广泛流行的计算机高级程序设计语言，它于 1973 年由美国贝尔实验室设计发布。由于 C 语言同时具备高级语言的优点和低级语言的效率，而且拥有很好的可移植性，使它成为程序员最喜欢的语言。

C 语言起源于对系统程序设计的深入研究和发展。C 语言是贝尔实验室的 Ken Thompson 和 Dennis Ritchie 等人开发的 UNIX 操作系统的“副产品”。Thompson 独自编写了 UNIX 操作系统的最初版本，与同时代的其他操作系统一样，UNIX 最初也是用汇编语言编写的，用汇编语言编写的程序往往难以调试和改进，UNIX 也不例外。Thompson 意识到需要用一种更加高级的编程语言来完成 UNIX 系统未来的开发，于是他设计了一种小型的 B 语言。B 语言是在 BCPL（Basic Combined Programming Language）的基础上开发的，而 BCPL 是在 CPL（Combined Programming）语言的基础上开发的，CPL 可以追溯到最早的语言之一 Algol60 语言。

1970 年，K. Thompson 用 B 语言在 PDP-7 机上实现了第一个实验性的 UNIX 操作系统。1972 年，贝尔实验室的 Dennis M. Ritchie 为克服 B 语言的诸多不足，在 B 语言的基础上重新设计了一种语言，由于是 B 语言的后继，故称为 C 语言。1973 年，贝尔实验室的 K. Thompson 和 Dennis M. Ritchie 合作，首先用 C 语言重新改写了 UNIX 操作系统，在当时的 PDP-11 计算机上运行。此后，C 语言作为 UNIX 操作系统上标准的系统开发语言，伴随着 UNIX 操作系统的发展，C 语言越来越广泛地被人们接受和应用。

1977 年，出现了独立于具体机器的 C 语言编译版本。由于 C 语言的独立和推广，也推动了 UNIX 操作系统在各种机器上的迅速实现。

1978 年，美国电话电报公司（AT&T）贝尔实验室正式发表了 C 语言。同时，由 B.W.Kernighan 和 D.M.Ritchie 合著了著名的《The C Programming Language》一书，简称为《K&R》，也有人称之为《K&R》标准。此书中介绍的 C 语言成为后来广泛使用的 C 语言版本基础，它被称为标准 C 语言。

1983 年，美国国家标准化协会（ANSI）根据各种 C 语言版本对 C 的扩充和发展，颁布了 C 语言的新标准 ANSI C。ANSI C 比标准 C 有了很大的扩充和发展。

1987 年，美国国家标准化协会在综合各种 C 语言版本的基础上，又颁布新标准，为了与标准 ANSI C 区别，所以称为 87 ANSI C。1990 年，国际标准化组织 ISO 接受了 87 ANSI C 作为 ISO C 的标准。这是目前功能最完善、性能最优良的 C 新版本。目前流行的 C 编译系统都是以它为基础的。

1989 年，X3J11 提出了一个报告[ANSI 89]，后来这个标准被 ISO 接受为 ISO/IEC 9899—1990。

1990 年，国际标准化组织ISO（International Organization for Standards）接受 89 ANSI C 作为 ISO C 的标准（ISO 9899—1990）。1994 年，ISO 修订了 C 语言的标准。

1995 年，ISO 对 C90 做了一些修订，即 1995 基准增补 1（ISO/IEC/9899/AMD1:1995）。1999 年，ISO 再次对 C 语言标准进行了修订，在保留原来 C 语言特征的基础上，针对实际编写应用程序的需要，增加了一些功能，尤其是完善了 C++语言中的一些功能，并将其命名为 ISO/IEC9899:1999。

2001 年和 2004 年先后进行了两次技术修正。目前流行的 C 语言编译系统大多是以 ANSI C 为基础进行开发的，但不同版本的 C 编译系统所实现的语言功能和语法规则又略有差别。

2011 年 12 月，ISO 正式公布 C 语言新的国际标准草案：ISO/IEC 9899:2011。

新的标准修订了 C11 版本，提高了对 C++的兼容性，并将新的特性增加到 C 语言中。新功能包括支持多线程，基于 ISO/IEC TR 19769:2004 规范下支持 Unicode，提供更多用于查询浮点数类型特性的宏定义和静态声明功能。

1.4　C 语言的特点

C 语言发展如此迅速，而且成为最受欢迎的语言之一，主要是因为它具有强大的功能。许多著名的系统软件，如 DBASE Ⅲ PLUS、DBASE Ⅳ 都是由 C 语言编写的。用 C 语言加上一些汇编语言子程序，就更能显示 C 语言的优势了，像 PC-DOS 、WORDSTAR 等就是用这种方法编写的。从语言体系和结构上讲，它是结构化程序设计语言；但从用户应用、实现难易程度、程序设计风格等角度来看，C 语言的特点又是多方面的。

1．C 是中级语言

C 语言通常称为中级计算机语言，它把高级语言的基本结构和语句与低级语言的实用性结合起来。中级语言并没有贬义，不意味着它功能差、难以使用，或者比 BASIC、Pascal 那样的高级语言原始，也不意味着它与汇编语言相似，会给使用者带来类似的麻烦。作为中级语言，C 允许对位、字节和地址这些计算机功能中的基本成分进行操作。C 语言可以实现汇编语言的大部分功能，可以直接操作计算机硬件如寄存器，各种外设 I/O 端口等。C 语言的指针可以直接访问内存物理地址，类似汇编语言的位操作可以方便地检查系统硬件的状态等。

所有的高级语言都支持数据类型的概念。一个数据类型定义了一个变量的取值范围和在其上

操作的一组运算。常见的数据类型是整型、字符型和实数型。虽然 C 语言有五种基本数据类型，但与 Pascal 或 Ada 相比，它却不是强类型语言。C 程序允许几乎所有的类型转换。例如，字符型和整型数据能够自由地混合在大多数表达式中进行运算。这在强类型高级语言中是不允许的。

C 语言的另一个重要特点是，它仅有 32 个关键字，这些关键字就是构成 C 语言的命令。和 IBM PC 的 BASIC 相比，后者包含的关键字达 159 个之多。

2．C 是结构式语言

虽然从严格的学术观点上看，C 语言是块结构（block-structured）语言，但是它还是常被称为结构化语言。这是因为它在结构上类似于 ALGOL、Pascal 和 Modula-2（从技术上讲，块结构语言允许在过程和函数中定义过程或函数。用这种方法，全局和局部的概念可以通过“作用域”规则加以扩展，“作用域”管理变量和过程的“可见性”。因为 C 语言不允许在函数中定义函数，所以不能称之为通常意义上的块结构语言）。

结构化语言的显著特征是代码和数据的分离。这种语言能够把执行某个特殊任务的指令和数据从程序的其余部分分离出去、隐藏起来。获得隔离的一个方法是调用使用局部（临时）变量的子程序。通过使用局部变量，我们能够写出对程序其他部分没有副作用的子程序。这使得编写共享代码段的程序变得十分简单。如果开发了一些分离很好的函数，在引用时，我们仅需要知道函数做什么，不必知道它如何做。切记：过度使用全局变量（可以被全部程序访问的变量）会由于意外的副作用而在程序中引入错误。

结构化语言比非结构化语言更易于程序设计，用结构化语言编写的程序的清晰性使得它们更易于维护。这已是人们普遍接受的观点了。C 语言的主要结构成分是函数 C 的独立子程序。

在 C 语言中，函数是一种构件（程序块），是完成程序功能的基本构件。函数允许一个程序的诸多任务被分别定义和编码，使程序模块化。可以确信，一个好的函数不仅能正确工作且不会对程序的其他部分产生副作用。

3．C 语言功能齐全

C 语言具有各种各样的数据类型，包括整型、实型、字符型、数组类型、指针类型、结构体类型、联合体类型等，因此，便于实现各种复杂的数据结构，如线性表、栈、队列、树和图等的运算。

并引入了指针概念，可使程序效率更高。另外，C 语言也具有强大的图形功能，支持多种显示器和驱动器。而且，计算功能、逻辑判断功能也比较强大，可以实现决策目的。

指针是 C 语言的一大特色，可以说是 C 语言优于其他高级语言的一个重要原因。就是因为它有指针，所以可以直接进行靠近硬件的操作，但是对 C 语言的指针操作不做保护，也给它带来了很多不安全的因素。C++在这方面做了改进，在保留了指针操作的同时又增强了安全性，受到了一些用户的支持。但是，由于这些改进增加语言的复杂度，也为另一部分人诟病。Java则吸取了C++的教训，取消了指针操作，也取消了 C++改进中一些备受争议的地方，在安全性和适合性方面均取得良好的效果，但其本身解释在虚拟机中运行，运行效率低于 C++/C。一般而言，C、C++、Java 被视为同一系的语言，它们长期占据着程序使用榜的前三名。

4．C 语言适用范围大

C 语言还有一个突出的优点是适合于多种操作系统，如 DOS、UNIX，也适用于多种机型。C 语言拥有一个庞大的数据类型和运算符集合，这个集合使得 C 语言具有强大的表达能力，可以用来编写各种系统软件和应用软件。

5．程序生成的目标代码质量高、程序运行效率高

试验表明，C 语言源程序生成的运行程序的效率仅比汇编程序的效率低 10%～20%，但 C 语言的编程速度快，程序可读性好，易于调试、修改和移植，这些优点是汇编语言所无法比拟的。

6．可移植性好

C 语言程序非常容易移植。可移植性表示为某种计算机写的软件可以用到另一种机器上去。举例来说，如果为苹果机写的一个程序能够方便地改为可以在 IBM PC 上运行的程序，则称为是可移植的。C 语言在某种计算机系统上编制的程序，只需少量改动，甚至不做修改就能移植到各种不同型号的计算机上。

7．C 语言存在的不足之处

编程自由度大，编译程序查错纠错能力有限，给不熟练的程序员带来一定困难；C 语言的理论研究及标准化工作也有待推进和完善。

高级语言接近人类语言和人们习惯使用的数学用语，不依赖于具体的机器，有严格的语法规则。相对于机器语言和汇编语言，高级语言易学易用，所编写的程序易读易改，通用性更强。C 语言把高级语言的基本结构与低级语言的高效实用性很好地结合起来，不失为一个出色而有效的现代通用程序设计语言。C 语言在计算机程序语言研究方面具有一定价值，由它引出了许多后继语言。C 语言已经成为世界上广泛流行的计算机高级程序设计语言，由于 C 语言同时具备高级语言的优点和低级语言的效率，而且拥有很好的可移植性，因此它成为程序员最喜欢的编程语言之一。C 语言以功能强大、数据结构丰富、目标代码质量高、程序运行效率高、可移植性好等特点成为众多程序员学习程序设计的首选语言，对整个计算机工业和应用的发展都起了很重要的推动作用。

1.5　C 程序的基本组成

1.简单 C 程序介绍

用 C 语言的语句编写的程序称为 C 语言程序（简称 C 程序）或 C 语言源程序。本节通过几个简单的 C 程序实例，介绍 C 程序的基本组成和结构，使读者对 C 语言和 C 程序的特性有初步的了解。

【例 1.1】在屏幕上输出：Hello,World!。

C 语言源程序如下：

```
//example1.1 The first C Program
#include <stdio.h>      //头文件
void main()
{
     printf("Hello,World!\n");
}
```

图 1-4　例 1.1 运行结果

运行结果如图 1-4 所示。

分析：

①void main 表示“主函数”。每个 C 程序都必须有且只能有一个 main 函数，它是每一个 C 程序执行的起始点（入口点）。void 表示该函数没有返回值。

②用{}括起来的是主函数 main 的函数体。main 函数中的所有操作（或：语句）都在这一对{}之间，即 main 函数的所有操作都在 main 函数体中。

③本程序的“主函数”main 中只有一条函数调用语句，printf()是 C 语言的库函数，用于程序中数据的输出（显示在屏幕上），本例是将一个字符串“Hello,World!\n”的内容输出，即在屏幕上显示：Hello,World!。

④每条语句用“;”号结束语句。

⑤头文件是每一个 C 程序必不可少的组成部分，因为一个 C 程序至少要包含输入或输出函数，而这些函数将被存放在以“.h”为扩展名的头文件中，比如本例的 printf 函数就包含在头文件“stdio.h”中。

⑥/*example1.1 The first C Program*/为注释，注释只是为了改善程序的可读性（提示、解释作用），在编译、运行时不起作用（编译时会跳过注释，目标代码中不会包含注释）。注释可以放在程序任何位置，并允许占用多行，只是需要注意“/*”、“*/”匹配，不要嵌套注释。

注释与软件的文档同等重要，要养成书写注释的良好习惯，这对软件的维护相当重要。因为程序是要给别人看的，自己也许还会看自己几年前编制的程序，清晰的注释有助于读者理解程序、算法的思路。

在软件开发过程中，还可以用注释帮助调试程序，即暂时屏蔽一些不需要运行的语句，以后可以方便地恢复。

【例 1.2】计算两数之和，并输出结果。

```
main()                    /*计算两数之和*/
{
  int a,b,sum;            /*定义变量*/
  a=11;
  b=22;                   /*以下 3 行为 C 语句*/
  sum=a+b;
  printf("sum=%d\n",sum);
}
```

运行结果如图 1-5 所示。

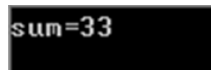


图 1-5　例 1.2 运行结果

分析：

①同样，此程序也必须包含一个 main 函数作为程序执行的起点。{}之间为 main 函数的函数体，main 函数所有操作均在 main 函数体中。

②int a,b,sum;是变量声明。声明了三个具有整数类型的变量 a,b,sum。C 语言的变量必须先声明再使用。

③a=11;b=22;是两条赋值语句。将整数 11 赋给整型变量 a，将整数 22 赋给整型变量 b。a,b 两个变量的值分别为 11, 22。注意这是两条赋值语句，每条语句均用“;”结束。

也可以将两条语句写成两行：

```
a=11;
b=22;
```

由此可见，C 程序的书写可以相当随意，但是为了保证容易阅读要遵循一定的规范。

④sum=a+b;是将 a,b 两变量内容相加，然后将结果赋值给整型变量 sum。此时，sum 的内容为 33。

⑤printf("sum=%d\n", sum);是调用库函数输出 sum 的结果。%d 为格式控制，表示 sum 的值以十进制整数形式输出。程序运行后，输出（显示）：sum=33。

【例 1.3】求两个数的较大值。

```
int max(int a,int b);                /*函数说明*/
main()                               /*主函数*/
{
  int x,y,z;                         /*变量说明*/
int max(int a,int b);                /*函数说明*/
  printf("input two numbers:\n");
scanf("%d%d",&x,&y);                 /*输入 x,y 值*/
  z=max(x,y);                        /*调用 max 函数*/
  printf("maxmum=%d",z);             /*输出*/
 }
int max(int a,int b)                 /*定义 max 函数*/
{
  if(a>b)return a;else return b;     /*把结果返回主调函数*/
 }
```

运行结果如图 1-6 所示。

```
input two numbers:
4 5
maxmum=5
```

图 1-6　例 1.3 运行结果

分析：程序由两个函数组成，即主函数和 max 函数。函数之间是并列关系，可从主函数中调用其他函数。max 函数的功能是比较两个数，然后把较大的数返回给主函数。max 函数是一个用户自定义函数。因此，在主函数中要给出说明（程序第三行）。可见，在程序的说明部分中，不仅可以有变量说明，还可以有函数说明。关于函数的详细内容将在以后章节中详细介绍。在程序的每行后用/*和*/括起来的内容为注释部分，程序不执行注释部分。

程序的执行过程是，首先在屏幕上显示提示串，请用户输入两个数，回车后由 scanf 函数语句接收这两个数送入变量 x,y 中，然后调用 max 函数，并把 x,y 的值传送给 max 函数的参数 a,b。在 max 函数中比较 a,b 的大小，把大者返回给主函数的变量 z，最后在屏幕上输出 z 的值。

2．C 程序的基本组成

综合上述三个例子，我们对 C 程序的基本组成和程序结构有了一个初步了解。

①C 程序由函数构成（函数是 C 程序的基本单位），所有的 C 程序都由一个或多个函数构成。其中，main 函数必须有且只能有一个。

②被调用的函数可以是系统提供的库函数，也可以是用户根据需要自己设计编写的函数。程序的全部工作由各个函数完成。编写 C 程序就是编写一个个函数。

③main 函数（主函数）是每个程序执行的起始点。无论 main 函数在程序中的哪个位置，一个 C 程序总是从 main 函数开始执行，也是从主函数结束。

④一个函数由函数首部和函数体两部分组成：

函数类型 函数名(参数类型及参数名表)

函数体：函数首部下方用一对{}括起来的部分。函数体一般包括声明和执行两部分。

例如函数 max:

```
函数类型   函数名称   参数类型    参数名   参数类型   参数名
 ↓          ↓          ↓         ↓        ↓        ↓
int         max      (int       x,      int       y)   /*函数首部*/
{   /*函数开始*/
    int z;  /*声明部分，定义变量*/
    if(x>y)
```

```
    z=x;
        else
    z=y;    /*函数体，执行部分*/
        return z;
    }   /*函数结束*/
```

⑤C 程序书写格式自由，一行可以写几个语句，一个语句也可以写在多行上。每条语句的最后必须有一个分号“;”表示语句的结束。

⑥可以使用/* */对 C 程序中的任何部分做注释。注释可以提高程序的可读性，使用注释是编程人员的良好习惯。实践中，编写好的程序往往需要修改、完善，事实上没有一个应用程序是不需要修改、完善的。很多人会发现自己编写的程序在经历了一些时间以后，由于缺乏必要的文档、必要的注释，最后连自己都很难再读懂，需要花费大量时间重新思考、理解原来的程序，这浪费了大量的时间。如果一开始编程时就对程序做必要的注释，虽然刚开始麻烦一些，但日后可以节省大量的时间。一个实际的系统往往是多人合作开发的，程序文档、注释是其中重要的交流工具。

⑦C 语言本身不提供输入/输出语句，输入/输出操作是通过调用库函数（scanf，printf 等）完成。

输入/输出操作涉及具体计算机硬件，把输入/输出操作放在函数中处理，可以简化 C 语言和 C 的编译系统，便于 C 语言在各种计算机上实现。不同的计算机系统需要对函数库中的函数做不同的处理，以便实现同样或类似的功能。

3．C 语言基本语法成分

①C 程序由函数构成（C 是函数式的语言，函数是 C 程序的基本单位）。

- 一个 C 源程序至少包含一个 main 函数，也可以包含一个 main 函数和若干个其他函数。函数是 C 程序的基本单位。
- 被调用的函数可以是系统提供的库函数，也可以是用户根据需要自己编写设计的函数。C 是函数式的语言，程序的全部工作都由各个函数完成。编写 C 程序就是编写一个个函数。
- C 函数库非常丰富，ANSI C 提供 100 多个库函数，Turbo C 提供 300 多个库函数。

②main 函数（主函数）是每个程序执行的起始点。

一个 C 程序总是从 main 函数开始执行，而不论 main 函数在程序中的哪个位置。可以将 main 函数放在整个程序的最前面，也可以放在整个程序的最后，或者放在其他函数之间。

③源程序中可以有预处理命令（例 1.1 中的 include 命令仅为其中的一种），预处理命令通常应放在源文件或源程序的最前面。

④C 语言字符集：C 语言字符集由字母、数字、各种符号组成（在字符串常量和注释中还可以使用汉字等其他图形符号），由字符集中的字符可以构成 C 语言的语法成分（如标识符、关键字、运算符等）。

- 字母：A～Z，a～z。
- 数字：0～9。
- 各种符号：空白符、标点符号、特殊字符等。

空白符：空格，制表符（跳格），换行符（空行）的总称。空白符除了在字符、字符串中有意义外，编译系统忽略其他位置的空白。空白符在程序中只是起到间隔作用。在程序的恰当位置使用空白将使程序更加清晰，增强程序的可读性。

标点符号、特殊字符：

```
!   #   %   ^   &   +   -   *   /   =   ～  <   >   \   |   .   ,   ;   :   ?
'   "   (   )   [   ]   {   }
```

以上字符集构成了 C 语言基本的语法元素。

⑤标识符：用来标识变量名、符号常量名、函数名、数组名、类型名等的有效字符序列。简单地说，标识符就是一个名字。

C 语言标识符命名规则：

标识符只能由字母、数字和下画线三种字符组成，且第一个字符必须为字母或下画线。

例如：

合法的标识符：sum，f1，average，_total，Class，day，stu_name，lotus_1_2_3。

不合法的标识符：M.D.John，$123，#45，7days，x-y，a+b。

注意：

- C 语言中大小写字母是两个不同的字符。例如：sum 不同 Sum，BOOK 不同 book。
- 标识符不能与“关键字”同名，也不与系统预先定义的“标准标识符”（如 main、printf 等）同名。
- 长度在一定范围内。各编译系统都有自己的规定和限制，一般环境允许取 32 个字符。
- 建议：标识符命名应当有一定的意义，做到见名知义。例如：sum 代表和，aver 代表平均值。

⑥关键字：C 语言规定的具有特定意义的字符串，用户只能按其预先规定的意义来使用它，不能改变其意义。C 语言可以使用以下 32 个关键字。

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

⑦运算符：运算符将常量、变量、函数连接起来组成表达式，表示各种运算。运算符可以由一个或多个字符组成，各运算符的运算规则将在后续的学习中陆续介绍。

⑧分隔符：逗号、空格等。起分隔、间隔作用。

⑨数据：程序要加工处理的对象。一个程序的主要任务就是对数据进行处理，一般程序中包括说明数据、输入数据、加工数据、输出数据等。

⑩C 语言本身不提供输入/输出语句，输入/输出的操作是通过调用库函数（scanf，printf）完成。

输入/输出操作涉及具体计算机硬件，把输入/输出操作放在函数中处理，可以简化 C 语言和 C 语言的编译系统，便于 C 语言在各种计算机上实现。不同的计算机系统需要对函数库中的函数做不同的处理，以便实现同样或类似的功能。

不同的计算机系统除了提供函数库中的标准函数外，还按照硬件的情况提供一些专门的函数。因此，不同计算机系统提供的函数数量、功能会有一定差异。

4．C 程序书写注意事项

①C 程序书写格式自由。

- 一行可以写几个语句，一个语句也可以写在多行上。
- C 程序没有行号，也没有 FORTRAN、COBOL 那样严格规定书写格式（语句必须从某一列开始）。
- 每条语句的最后必须有一个分号“;”表示语句的结束。

②可以使用/*…*/（可换行写）或者//…（不可换行写）对 C 程序中的任何部分做注释，注释可以提高程序可读性，使用注释是编程人员的良好习惯。

- 代码注释的作用一直都被程序员们广泛讨论，很多人认为注释不是必要的，写注释是因为

代码可读性太差。实际上，编写程序比阅读程序要容易，代码的可读性显得尤为重要，所以程序员必须通过一些技术来提高程序的可读性，其中一点就包括代码注释。同一行的注释不要写得很长，更不要为了注释而注释，不要重复读者已经知道的内容，应该把长注释放在代码上面，短注释放在代码后面。

- 一个实际的系统往往是多人合作开发的，程序文档、注释是其中重要的交流工具。

③从书写清晰，便于阅读、理解和维护的角度出发，在书写程序时一般应遵循以下规则：

- 一个说明或一个语句占一行。
- 用{ }括起来的部分，通常表示程序的某一层次结构。{ }一般与该结构语句的第一个字母对齐，并单独占一行。
- 低一层次的语句或说明可比高一层次的语句或说明缩进若干格后书写，以便看起来更加清晰，增加程序的可读性。

1.6 C 语言的上机执行过程

编写出 C 程序仅仅是程序设计工作中的一个环节，写出来的程序需要在计算机上进行调试运行，直到得到正确的运行结果为止。

1.6.1 C 程序的开发过程

开发一个 C 程序一般要经过四个步骤，即编辑、编译、连接和运行，才能得到程序的运行结果。下面分别说明上机执行过程。

1．编辑

编辑是 C 语言源程序的输入和修改，并以文本文件的形式存放在磁盘上。其标识为：“文件名.C”。其中，文件名是由用户指定的符合操作系统文件命名规则的任意字符组合，扩展名要求为“.C”，表示是 C 语言源程序。例如 file1.c、t.c 等。用于编辑源程序的软件是编辑程序。编辑程序是提供给用户书写程序的软件环境，可用来输入和修改源程序。

2．编译

编译是把 C 语言源程序翻译成用二进制指令表示的目标文件。编译过程由编译系统提供的编译程序完成。编译程序自动对源程序进行句法和语法检查，当发现错误时，将错误的类型和所在的位置显示出来，提供给用户，以帮助用户修改源程序中的错误。如果未发现句法和语法错误，就自动形成目标代码并对目标代码进行优化后生成目标文件。目标程序的文件标识是：“文件名.obj”，扩展名“.obj”是目标程序的文件类型标识。

3．连接

连接过程是连接程序（也称链接程序或装配程序）将目标程序、库函数或其他目标程序连接装配成可执行程序。可执行程序的文件名为：“文件名.exe”，扩展名“.exe”是可执行程序的文件类型标识。

4．运行

运行程序是指将可执行程序投入运行，得到程序处理的结果。如果程序运行结果不正确，可重新回到第一步，重新对程序进行编辑修改、编译和运行。

与编译、连接不同，运行可以脱离语言处理环境。因为它是对一个可执行程序进行操作，与 C 语言源程序本身已经没有联系，所以可以在语言开发环境下运行，也可直接在操作系统下运行。

以上四个步骤的执行过程如图 1-7 所示。其中，图中虚线框内是 C 集成开发系统提供的语言处理程序和 C 标准库函数。

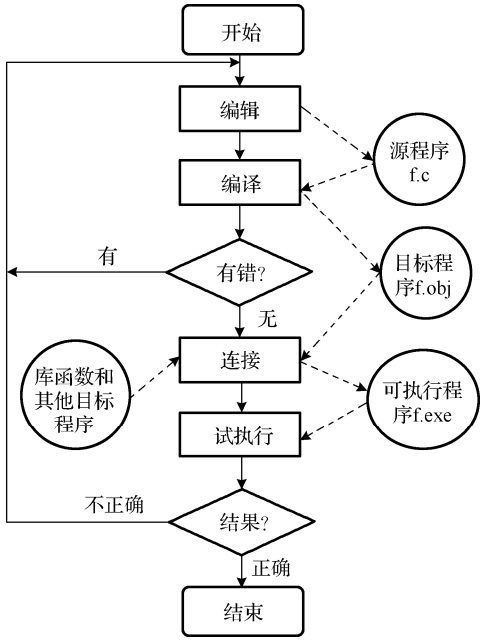


图 1-7　开发 C 程序的步骤

1.6.2　Visual C++6.0 开发环境及程序测试与调试

1．Visual C++6.0 开发环境

Visual C++6.0 是 Microsoft 公司推出的运行在 Windows 操作系统中的交互式、可视化集成开发软件，集程序的编辑、编译、连接、调试等功能于一体，为编程人员提供了一个既完整又方便的开发平台，它不仅支持 C++语言，也支持 C 语言，成为常用的 C++语言和 C 语言的开发环境。

Visual C++是一个集成的语言开发环境，首先必须在计算机系统中安装，安装完毕后，选择 Windows 的“开始”菜单中的“Microsoft Visual Studio 6.0 Microsoft”|“Visual C++6.0”命令，启动 Visual C++6.0；也可在桌面建立 Visual C++6.0 的快捷方式图标，双击即可启动；或者直接双击一个 C 语言或 C++语言的源程序，启动 Visual C++6.0。下面简单介绍在 VC 环境下，开发 C 语言源程序的过程。

2．C 语言源程序开发过程

1）编辑 C 源程序代码

启动 Visual C++6.0（简称 VC）集成开发环境，在 VC 主窗口（如图 1-8 所示）中选择“File|New”菜单命令，弹出“New（新建）”对话框。选择“New”对话框上的“File”选项卡，并选择“C++ Source File”选项，建立 C 语言源程序，如图 1-9 所示。

在对话框右侧的“Location（位置）”文本框中输入或选择新文件的存储位置。

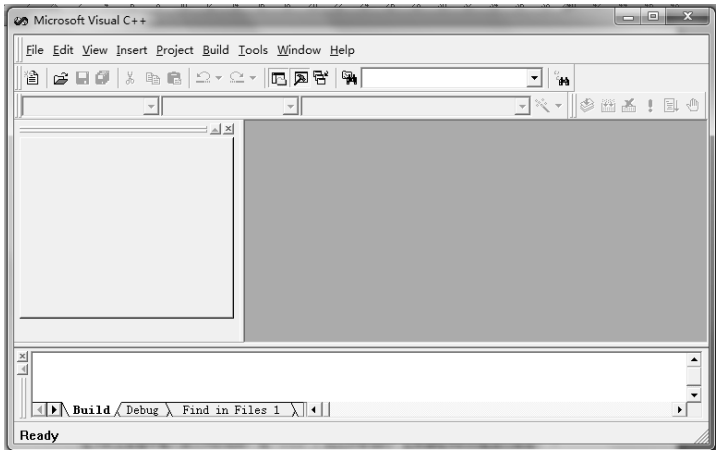


图 1-8　VC 主窗口

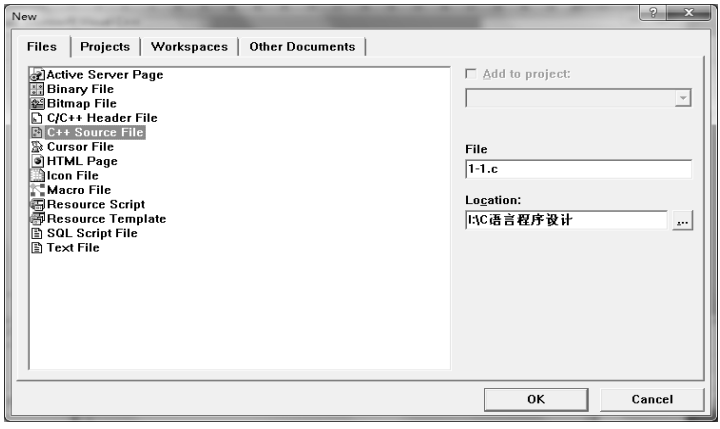


图 1-9 “New（新建）”对话框

在“File”文本框中输入新文件的名称，如“1-1.c”。对文件命名时，应根据程序的功能进行命名。

单击“OK”按钮，回到 VC 主窗口，在编辑窗口看到光标闪烁，此时可输入和修改源程序，如图 1-10 所示。

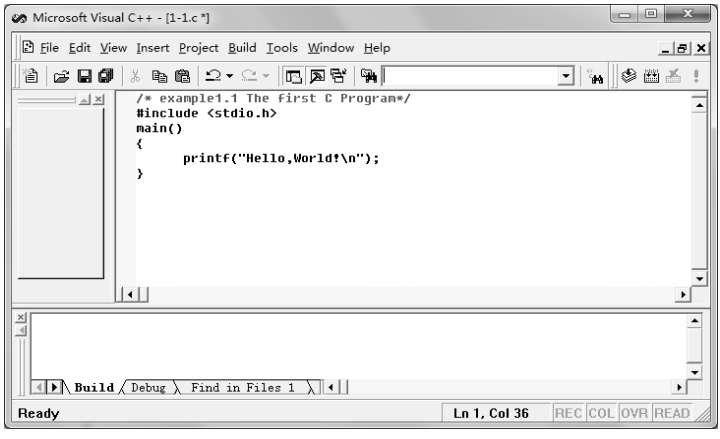


图 1-10　C 源程序编辑窗口

选择“File|Save”菜单命令，或单击工具栏中的“保存”按钮或按 Ctrl+S 组合键保存源程序文件。

2）编译、连接和运行程序

在 VC++6.0 中，文件是在项目的管理之下，而项目则在工作区的管理之下，在编辑 C 源程序代码时没有进行“新建工作区，在工作区下建立项目，在项目下建立 C 源程序代码”的过程，在进行 C 源程序代码编译时，可为源代码建立默认的工作区和项目。

（1）编译。单击工具栏上的“Compile”按钮（ ）或选择“Build”菜单中的“Compile 1-1.c（编译 1-1.c）”命令，编译系统对文件 1-1.c 进行编译，弹出如图 1-11 所示的对话框，单击“是”按钮，表示同意由 VC 系统建立一个默认的项目工作区和一个工程，并对源程序进行编译；若单击“否”按钮，将不会对源程序进行编译。

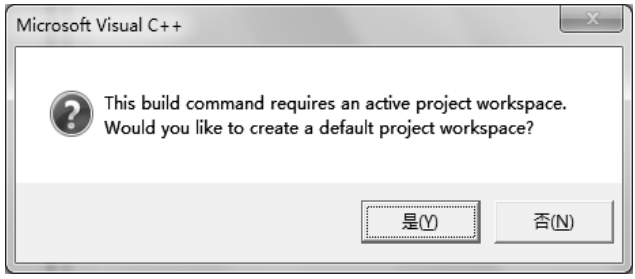


图 1-11　创建工作区对话框

若编译成功，则在 VC 输出窗口显示“0 error(s),0 warning(s)”，表示没有致命错误（error），也没有警告（warning），编译系统生成一个目标文件 1-1.obj，之后可以进行程序的连接与执行，VC 运行编译后的 VC 窗口如图 1-12 所示。

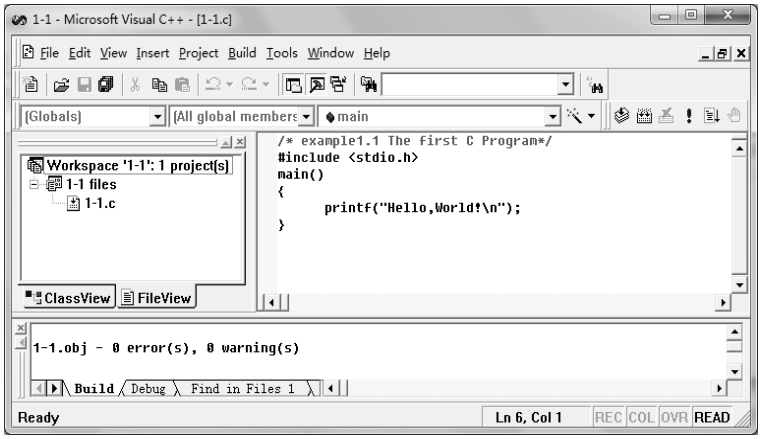


图 1-12　编译后的 VC 窗口

若编译有错，则在输出窗口显示错误信息。按 F4 键或双击错误提示行，在源程序出错行左侧出现标记，并且光标定位到此行。此时，应检查标记所在行（可能会是标记前一行或多行）的程序代码，找出错误的原因并改正，然后再编译，若出现错误，则再修改，直到编译通过为止。

（2）链接。单击工具栏上的“Build”按钮（ ），或选择“Build”菜单下的“Build 1-1.exe”命令，或按 F7 键对目标文件 1-1.obj 进行连接。若连接成功，则生成一个可执行文件 1-1.exe。

（3）运行。单击工具栏上的“Execute”按钮（!），或选择“Build”菜单下的“!Execute 1-1.exe”命令，或按 Ctrl+F5 组合键。运行后产生结果输出窗口。

程序执行完毕，单击“File”菜单下的“Save（保存）”命令，对源程序进行保存，也可以按 Ctrl+S 组合键来保存文件。

程序执行、保存完毕，单击“File”菜单下的“Close Workspace（关闭工作区）”命令，如图 1-13 所示，按以上步骤即可进行下一个程序的开发工作。

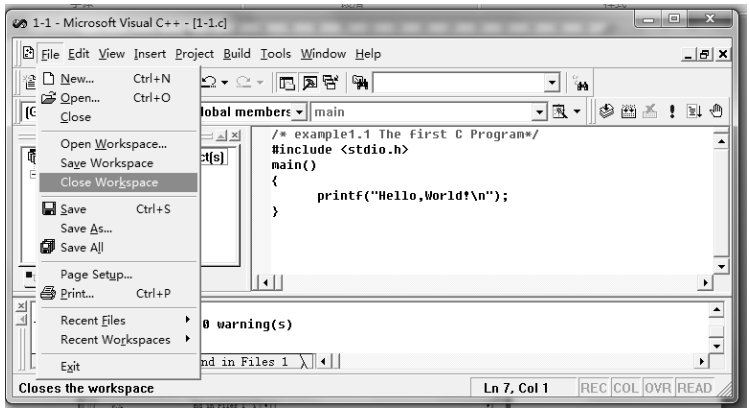


图 1-13　关闭工作区界面

工程工作区就像一个“容器”，由它来“盛放”相关工程的所有有关信息，创建默认工程工作区之后，系统将创建出一个相应的工作区文件（.dsw），用来存放与该工作区相关的信息；另外还将创建出的其他几个相关文件，即工程文件（.dsp）以及选择信息文件（.opt）等。

开发一个 C 语言源程序，也可以按照“创建工程工作区”|“创建工程”|“在工程下创建 C 源程序”的步骤进行，下面简略说明过程。

在如图 1-8 所示的 VC 主窗口下选择“file”|“new”菜单命令，出现如图 1-9 所示的对话框，选择“Projects”选项卡下的“Win32 Console Application”，在右侧的“Location”文本框和“Project name”文本框中填入工程存放在磁盘的位置（目录或文件夹位置）以及工程的名字，单击“OK”按钮进入如图 1-14 所示的界面。这个界面主要询问用户想要构成一个什么类型的工程。

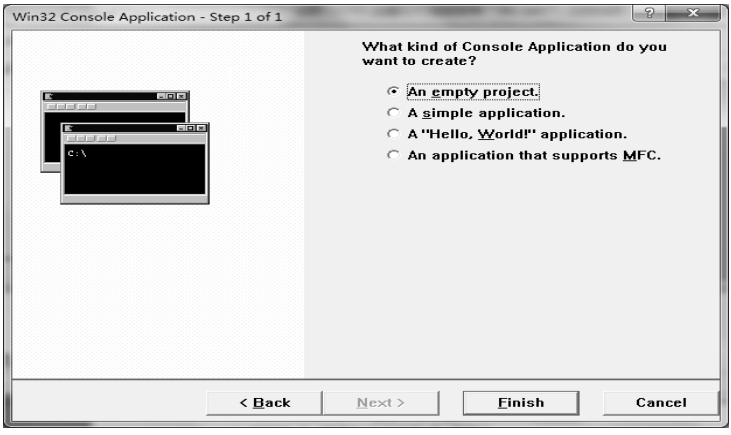


图 1-14　创建工程界面

选择“An empty project.”项将生成一个空的工程，工程内不包括任何东西。

选择“A simple application.”项将生成包含一个空的 main 函数和一个空的头文件的工程。选择“A"Hello，World!"application.”项与选择“A simple application.”项没有什么本质的区别，只是需要包含显示出“Hello，World!”字符串的输出语句。若选择“An application that supports MFC.”项，则可以利用 VC6 所提供的类库来进行编程。

选择“Project”菜单的“Add To Project”子菜单下的“new”项，在出现的对话框的“Files”选项卡中，选择“C++ Source File”项，在右中处的“File”文本框中为将要生成的文件取一个名字，取名为 1-1.c（见名知义），然后单击“OK”按钮，进入输入源程序的编辑窗口，输入 C 源程序代码，编译、连接、执行即可。

1.6.3　Turbo C 2.0 开发环境及程序测试与调试

1．Turbo C 简介

1）Turbo C 的产生与发展

Turbo C 是美国 Borland 公司的产品。Borland 公司是一家专门从事软件开发、研制的大公司。该公司相继推出了一套 Turbo 系列软件，如 Turbo BASIC、Turbo Pascal、Turbo Prolog，这些软件很受用户欢迎。该公司在 1987 年首次推出 Turbo C 1.0 产品，其中使用了全然一新的集成开发环境，即使用了一系列下拉式菜单，将文本编辑、程序编译、连接以及程序运行一体化，大大方便了程序的开发。1988 年，Borland 公司又推出 Turbo C 1.5 版本，增加了图形库和文本窗口函数库等，而 Turbo C 2.0 则是该公司在 1989 年出版的。Turbo C 2.0 在原来集成开发环境的基础上增加了查错功能，并可以在 Tiny 模式下直接生成.COM（数据、代码、堆栈处在同一 64KB 内存中）文件。还可对数学协处理器（支持 8087/80287/80387 等）进行仿真。

Borland 公司后来又推出了面向对象的程序软件包 Turbo C++，它继承、发展了 Turbo C 2.0 的集成开发环境，并包含了面向对象的基本思想和设计方法。

1991 年，为了适用 Microsoft 公司的 Windows 3.0 版本，Borland 公司又对 Turbo C++ 做了更新，即 Turbo C 的新一代产品 Borland C++也问世了。

2）Turbo C 2.0 基本配置要求

Turbo C 2.0 可运行于 IBM-PC 系列微机，包括 XT、AT 及 IBM 兼容机。此时要求 DOS 2.0 或更高版本支持，并至少需要 448KB 的 RAM，可在任何彩、单色 80 列监视器上运行。支持数学协处理器芯片，也可进行浮点仿真，这将加快程序的执行。

3）Turbo C 2.0 内容简介

Turbo C 2.0 包括以下主要文件：

INSTALL.EXE	安装程序文件
TC.EXE	集成编译
TCINST.EXE	集成开发环境的配置设置程序
TCHELP.TCH	帮助文件
THELP.COM	读取 TCHELP.TCH 的驻留程序
README	关于 Turbo C 的信息文件
TCCONFIG.EXE	配置文件转换程序
MAKE.EXE	项目管理工具
TCC.EXE	命令行编译

TLINK.EXE	Turbo C 系列连接器
TLIB.EXE	Turbo C 系列库管理工具
C0?.OBJ	不同模式启动代码
C?.LIB	不同模式运行库
GRAPHICS.LIB	图形库
EMU.LIB	8087 仿真库
FP87.LIB	8087 库
*.H	Turbo C 头文件
*.BGI	不同显示器图形驱动程序
*.C	Turbo C 例行程序（源文件）

其中，上面的?分别为:

T	Tiny（微型模式）
S	Small（小模式）
C	Compact（紧凑模式）
M	Medium（中型模式）
L	Large（大模式）
H	Huge（巨大模式）

2．Turbo C 环境下 C 程序的运行

为了使计算机能按照人们的意志工作，就要根据问题的要求，编写相应的程序。程序是一组计算机可以识别和执行的指令，每一条指令使计算机执行特定的操作。程序可以用高级语言或汇编语言编写，用高级语言或汇编语言编写的程序称为源程序。C 程序源程序的扩展名为“.c”。事实上，我们编写的程序，不管采用什么计算机语言，都是源程序，很少有人会用机器语言去编程！源程序不能直接在计算机上执行，需要用“编译程序”将源程序翻译为二进制形式的代码。

源程序经过“编译程序”翻译所得到的二进制代码称为目标程序。目标程序的扩展名为“.obj”。目标代码尽管已经是机器指令，但是还不能运行，因为目标程序还没有解决函数调用问题，需要将各个目标程序与库函数连接，才能形成完整的可执行的程序。

目标程序与库函数连接，形成的完整的可在操作系统下独立执行的程序称为可执行程序。可执行程序的扩展名为“.exe”（在 DOS/Windows 环境下）。

1）启动 TC（执行 TC .exe 文件）

进入 Turbo C 2.0 集成开发环境中后，屏幕上显示如图 1-15 所示。

图 1-15 Turbo C 2.0 集成开发环境

2）建立与编辑 C 源程序

（选择“File”菜单，或直接按 Alt+F 组合键）

在编辑窗内建立一个名为“NONAME.C”的空白文件；若选择“Load”或直接按 F3 键，随即会在屏幕上弹出一个“文件名”框，在框中输入要打开的文件名后，系统就依据文件名打开指定的文件（磁盘上应该保存有该文件，否则将以该文件名建立一个新文件）。如果在“文件名”框中输入的文件名是“*.C”，屏幕上就会出现一个列表框，其中列出了当前目录下所有扩展名为.C 的文件，使用光标移动键可以从中方便地选取所需打开的文件。

编辑好的程序可以作为磁盘文件保存到磁盘中，保存文件时可选择“File”菜单中的“Save”命令，或直接按 F2 键实现。

3）编译、连接 C 程序

编译源程序应选择“Compile（编译）”菜单中的“Compile to OBJ”命令，它可以对编辑窗内的源程序进行编译，生成与源程序同名的目标文件（扩展名为.OBJ）。

接着再选择“Link EXE file”命令，将当前编译生成的目标文件与库文件等连接，生成与目标文件同名的可执行文件（扩展名为.EXE）。

也还可以直接选择“Compile”菜单中的“Make EXE file”命令，将编译、连接操作过程合并，一次连续完成编译、连接过程，直接生成可执行文件。

4）运行程序

选择“Run”菜单中的“Run”命令，或直接按 Ctrl+F9 组合键，即可运行上述可执行文件程序。如果源程序正确，即可得到其执行结果。查看程序执行结果可直接按 Alt+F5 组合键。

5）查找与修改错误

在编译、连接过程中，C 编译系统会进行语法检查。如果发现程序中存在错误，就在信息窗口中显示有关出错信息。它不仅指出错误的位置，还说明错误的性质和原因。

按照窗口下部的提示，运用相应的命令对源程序进行编辑，修正程序错误，然后重复 2），…，直至得出正确结果。以上过程如图 1-16 所示。

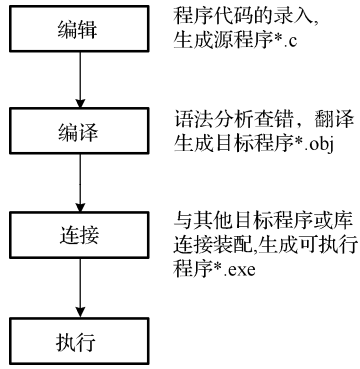


图 1-16　C 程序执行步骤

1.7　C 语言学习方法

1.7.1　为什么要学习 C 语言

对于不太可能成为专职程序员的大学生，为什么要学习 C 语言呢？

面向过程的程序设计语言在历史上出现过很多种，如 BASIC、Pascal、FORTRAN 等。因为种种原因，用的人越来越少，学习的人也就越来越少了。

进入 Windows 时代，出现了很多的面向对象的可视化编程语言，如 VC、Java、VB 等，但这些语言的重要基础之一还是面向过程的，初学者往往会先被其华丽便捷的界面设计所吸引，对最基础的面向过程程序设计方法却无法深入学习。学习 C 语言，将可以更准确地定位于计算机思维训练，更有效地理解计算机工作原理。

C 语言广泛应用于设备驱动程序，高性能、实时中间件，嵌入式领域，并发程序设计等，也是 IT 行业交流、笔试、面试时最常见的语言。

几乎没有不能用 C 语言实现的软件，也没有不支持 C 语言的系统，很多重要级的软件都是用 C 语言写成的。更重要的是很多流行语言、新生语言都借鉴了它的思想、语法，如 C++、Java、C#等。

正因为如此，在 2014 年 TIOBE 编程语言排行榜中，C 语言以 17.588%的受欢迎程度排列第一，而且几十年来一直排在前三名的行列，如图 1-17 所示。

2014年TIOBE编程语言排行榜

Mar 2014	Mar 2013	Change	Programming Language	Ratings	Change
1	2	↑	C	17.5355%	+0.89%
2	1	↓	Java	16.406%	-1.75%
3	3		Objective-C	12.143%	+1.91%
4	4		C++	6.313%	-2.80%
5	5		C#	5.572%	-1.02%
6	6		PHP	3.698%	-1.11%
7	7		(Visual)Basic	2.955%	-1.65%
8	8		Python	2.021%	-2.37%
9	11		JavaScript	1.899%	+0.53%

图 1-17　常见编程语言排行榜

1.7.2　如何学习 C 语言

C 语言相对其他程序设计语言而言，给程序员的发挥空间最大、运行效率最高，给了程序员无限制的自由，这些优点也正是它让 C 语言初学者又爱又恨的原因。如何才能真正地学好 C 语言呢？

（1）多练多读优秀代码，光看教材例题是肯定学不好 C 语言的，只有在熟练掌握例题、理解例题的基础上，多上机实践，积累丰富的调试程序经验，透过 C 语言窥探计算机底层原理，掌握基本的程序设计思维。

（2）多交流，与同学交流，与老师交流，通过网络与网友交流。

1.7.3　C 语言学习资源

因为学习 C 语言的人多，所以任何问题在网络上都可以找到解决方案。有许多的网站既拥有大量的经典案例，又具有很高的人气。以下网站是值得大家交流的好去处。

（1）www.csdn.net

全球最大的中文 IT 社区，分为多个板块，免费注册用户后，可以在相应的板块中咨询关于计算机的任何问题。

（2）www.cyuyan.com.cn

C 语言网，主要针对 C/C++用户的一个交流平台。

（3）http://post.baidu.com/f?kz=8618367

C 语言由浅入深的 100 例经典源程序源代码，每一个例子都值得认真学习。

本 章 小 结

本章介绍计算机语言、计算机语言的发展、程序和程序设计的基本概念；主要说明一个 C 语

言程序的基本结构，让初学者对 C 语言程序结构有大致的了解，最后说明 C 语言程序的开发执行过程和在 VC++6.0 和 Turbo C 2.0 集成开发环境中如何开发一个 C 语言程序。

习　题　1

一、单选题

1．一个 C 语言源程序由（　　）组成。

A．函数　　B．main 函数　　C．子程序　　D．过程

2．以下关于一个 C 语言源程序执行的叙述中，正确的是（　　）。

A．一个 C 语言源程序的执行总是从第一个函数开始，在最后一个函数中结束

B．一个 C 语言源程序的执行总是从 main 函数开始，在 main 函数中结束

C．一个 C 语言源程序的执行总是从 main 函数开始，在最后一个函数中结束

D．一个 C 语言源程序的执行总是从第一个函数开始，在 main 函数中结束

3．以下说法中正确的是（　　）。

A．一个 C 函数中只允许一对花括号

B．在 C 语言程序中，要调用的函数必须在 main()函数中定义

C．C 语言不提供输入/输出语句

D．C 语言程序中的 main()函数必须放在程序的开始部分

4．下列四组选项中，均不是 C 语言关键字的选项是（　　）。

A	B	C	D
define	gect	include	while
IF	char	scanf	go
type	printf	case	pow

5．以下选项中不合法的用户标识符是（　　）。

A．abc.c　　B．file　　C．main　　D．print

6．下面描述中，正确的是（　　）。

A．主函数中的花括号必须有，而子函数中的花括号是可有可无的

B．一个 C 程序行只能写一个语句

C．主函数是程序启动时唯一的入口

D．函数体包含了函数说明部分

7．下面描述中，不正确的是（　　）。

A．C 程序的函数体由一系列语句和注释组成

B．注释内容不能单独写在一行上

C．C 程序的函数说明部分包括对函数名、函数类型、形式参数等的定义和说明

D．scanf 和 printf 是标准库函数而不是输入和输出语句

二、填空题

1．C 程序是由_____构成的，一个 C 程序中至少包含_____。

2．C 程序注释是由_____和_____所界定的文字信息组成的。

3．函数体一般包括_____和_____。

4．为解决某一实际问题而设计的指令序列就是_____。

5．目标程序文件的扩展名是_____。

6．在程序设计中，把解决问题确定的方法和有限的步骤称为_____。

7．一个完整的 C 程序至少要有一个而且只能有一个_____函数。

8．开发一个 C 程序要经过编辑、_____、_____和运行 4 个步骤，才能得到运行结果。

9．程序连接过程是将目标程序、_____或其他目标程序连接装配成可执行文件。

10．因为源程序是_____类型的文件，所以它可以用具有文本编辑功能的任何编辑程序完成编辑。

三、编程题

1．在屏幕上打印图案：⊙_⊙

2．编写程序，从键盘输入两个整型数字，分别求这两个数字的和、差、积和商并输出结果。

3．编写程序，从键盘输入梯形的上底、下底和高，求该梯形的面积并输出结果。

第 2 章　C 语言基础

2.1　C 语言的数据类型

数据是对客观事物的符号表示，是所有能被输入到计算机中，且能被计算机处理的符号（数字、字符等）的集合，它是计算机操作对象的总称。

数据是程序处理的对象，一个数据必定属于某一种数据类型，不同的数据类型有不同的操作，也决定了数据的取值范围以及它们在计算机中的存储形式。C 语言的数据类型如图 2-1 所示。

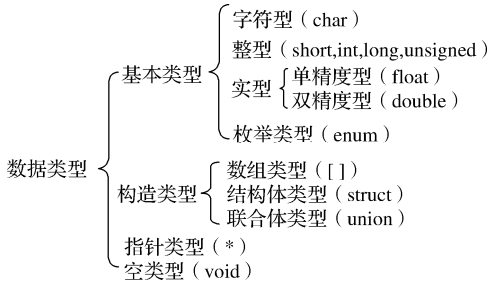


图 2-1　C 语言的数据类型

C 语言为每个类型定义了一个标识符，通常把它们称为类型名。例如整数型用 int 标识，字符型用 char 标识。一个类型名由一个或几个关键字组成。

C 语言的数据类型比其他一些程序语言要丰富，它有指针类型，还有构造其他多种数据类型的能力。例如，除了数组类型之外，C 语言还可以构造结构体类型、联合类型和枚举类型等多种数据类型。基本类型结构比较简单，构造类型一般是由其他的数据类型按照一定的规则构造而成的，结构比较复杂。指针类型是 C 语言中使用灵活、颇具特色的一种数据类型。

本章主要介绍基本数据类型中的整型和实型两大类，其他数据类型将在后续章节中详细介绍。

2.1.1　整型数据类型

要清楚整型数据，首先需要清楚计算机中数据的原码、反码、补码的概念。

1．整型数据编码

①原码

原码是指将一个数值的绝对值化为二进制数，再补齐或截取相应字节位后，将最高位用来表示符号，正数为 0，负数为 1 形成的二进制编码。例如：

```
10 的双字节原码：      0000 0000 0000 1010
-10 的双字节原码：     1000 0000 0000 1010
```

②反码

正数的反码与原码相同，负数的反码是将原码除符号位外各位逐一取反形成的二进制编码。例如：

-10 的双字节反码： 1111 1111 1111 0101

③补码

正数的补码与原码相同，负数的补码就是该数的反码+1 形成的二进制编码。例如：

-10 的双字节补码：1111 1111 1111 0110

同样，如果-10 用 4 个字节表示，则原码、反码、补码表示如下：

-10 的 4 个字节原码： 1000 0000 0000 0000 0000 0000 0000 1010
-10 的 4 个字节反码： 1111 1111 1111 1111 1111 1111 1111 0101
-10 的 4 个字节补码： 1111 1111 1111 1111 1111 1111 1111 0110

所以可以这样说，数据在计算机中存储的是补码。

由运算规则可以知道，对补码再求补码得到原码。

2．整型数据的表示

字符型数据在计算机中占一个字节，用 8 位二进制表示。当最高位用于表示符号时，数据表示范围为-27～27-1（-128～127）。由于字符型数据通常用于存储 ASCII 表中字符的 ASCII 值，所以称为字符型。如字符‘A’在计算机中存储的实际上是其 ASCII 值 65。

同理，短整型数据占 2 个字节，表示数的范围为-215～215-1（-32768～32767）。而长整型数据占 4 个字节，表示范围为-231～231-1（-2147483648～2147483647）。

C 语言中还可以将数据位全部用于表示数据大小，这种数据类型称为无符号型，表示数据的范围也相应发生了变化。整型数据的长度及数的表示范围如表 2-1 所示。

表 2-1 整型数据的长度及数的表示范围

类型说明符	分配字节数	数 的 范 围
int	2	-32768～32767 即-215～（215-1）
	4	-2147483648～2147483647 即-231～（231-1）
short [int]	2	-32768～32767 即–215～（215-1）
long [int]	4	-2147483648～2147483647 即-231～（231-1）
long long [int]	8	-9223372036854775808～9223372036854775807 即-263～（263-1）
unsigned [int]	2	0～65535 即 0～（216-1）
	4	0～4294967295 即 0～（232-1）
unsigned short [int]	2	0～65535 即 0～（216-1）
unsigned long [int]	4	0～4294967295 即 0～（232-1）
unsigned long long int	8	0～18446744073709551615 即 0～（264-1）

表 2-1 列出了不同编译系统给不同类型分配了不同的存储空间。例如，系统在编译时，分配给 int 型数据 2 个字节或 4 个字节，这由具体的 C 编译系统决定，例如 Turbo C 2.0 为每个整型数据分配 2 个字节，Visual C++为每个整型数据分配 4 个字节。编写程序需要注意，当整型变量的实际数值大于它所能容纳的最大数值时，就会发生“溢出”，但运行时并不报错。

3．整型数据的溢出

当进行整型数据计算时，若计算结果超出了该类数据表示的范围，这种情况称为数据溢出。先看示例。

【例 2.1】编写求两数和的 C 程序并上机运行。

C 语言源程序如下：

```
#include <stdio.h>
void main()                    /*求两数和主函数*/
{    short  a,b;               /*说明 a、b 为整型变量*/
     a=32767;                  /*为变量 a 赋值 32767*/
     b=a+2;                    /*将变量 a 的值+2 后赋给变量 b*/
     printf("b=%d\n",b);       /*输出变量 b 的值*/
}
```

预测结果：32769

运行结果如图 2-2 所示。

图 2-2　例 2.1 运行结果

验证结果：-32767

为什么会出现这种情况呢？

变量 a 的值二进制表示为：0111 1111 1111 1111

变量 b 的值二进制表示为：1000 0000 0000 0001

变量 b 的值首位为 1，表示为负数，而计算机中存储的都是补码，要知道负数的原码，必须对该补码求补。

变量 b 中数值的补码为：1000 0000 0000 0001

对补码求反码得：1111 1111 1111 1110

对反码+1 得原码为：1111 1111 1111 1111

这个数化为十进制正好是-32767。

对于这种问题，系统往往不给出错误提示，而要靠程序设计者正确使用内存空间来保证其正确性，所以对数据类型的使用要仔细，对运算结果的数量级要有基本估计。

2.1.2　实型数据类型

实型数据类型用于表示超过整型表示范围内的整数和带小数点的数，根据表示范围及精度要求的不同，分为单精度型和双精度型。实型数据类型的长度及数的表示范围如表 2-2 所示。

表 2-2　实型数据类型的长度及数的表示范围

实数类型	类型标识符	存储字节数	取值范围	有效位
单精度型	float	4 字节	-3.4×10^{38}～3.4×10^{38}	7
双精度型	double	8 字节	-1.7×10^{308}～1.7×10^{308}	16

实数是以指数形式存储的，对任何形式的实数，均转化成指数形式，如 345.68 转化成 0.34568e3，12.45e3 则转化成 0.1245e5 的形式存储。

例如：以微机中存储 float 类型数据为例，在内存中占据 4 个字节，即 32 位，这 32 位分别存放该数据的符号（数据正负符号）、规范化的尾数（小数部分）、阶符（指数正负符号）和阶码（指数）。例如：0.123456e-2 的存放形式可用图 2-3 示意。实际上，计算机中存放的是二进制数，这里仅用十进制数说明其存放形式。对于尾数和阶码各占多少二进制位，标准 C 并无具体规定，由各编译系统自行确定。

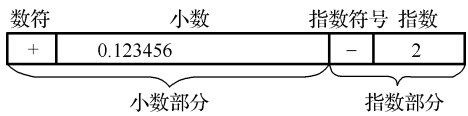


图 2-3　实型数据在计算机中的存放形式

对于实型数据来说，在计算过程中可能会存在一定的误差。

【例 2.2】输出实型数据 a,b 的值。

C 语言源程序如下：

```
#include <stdio.h>
void  main()
{   float a;                          /*定义变量 a 为单精度型*/
    double b;                         /*定义变量 b 为双精度型*/
    a=-12345.633;
    b=-0.12345678912345673 99e15;
    printf("a=%f,b=%f\n",a,b);        /*输出变量 a、b 的值*/
}
```

程序为单精度变量 a 和双精度变量 b 分别赋值，并不经过任何运算就直接输出变量 a，b 的值。理想结果应该是照原样输出，即

```
a=-12345.633，b=-0.123456789123456 7399e15
```

运行结果如图 2-4 所示。

```
a=12345.678711,b=1234567891234 56.800000
```

图 2-4　例 2.2 运行结果

分析：因为程序中变量 a 为单精度型，只有能存储 7 位有效数字，所以输出的前 8 位是准确的，第 9 位以后的数字“711”是无意义的。变量 b 为双精度型，可以存储 15～16 位有效位，所以输出的前 15 位是准确的，第 16 位以后的数字“800000”是无意义的。由此可见，由于机器存储的限制，使用实型数据会产生一些误差，要注意实型数据的有效位，合理使用不同的类型，尽可能减少误差。

2.2 常　　量

常量是指在程序运行过程中，其值不能被改变的量，实际上常量即常数。常量是 C 语言中合适的基本数据对象之一，包括字符常量、短整型常量、整型常量、单精度常量、双精度常量、字符串常量等。

2.2.1 整型常量

整型常量就是整型常数，也称为直接常量。在计算机中，数据以二进制补码形式进行存储。在 C 语言中为了方便表示及使用数据，我们采用十进制、八进制、十六进制这三种形式。编译系统会自动将各种进制数据转换为二进制的形式进行存储。

十进制整型常量与我们日常生活中所使用的形式基本相同，数码由 0～9 组成，数字前可以带正负号。

八进制整型常量必须以数字 0 开头，数码由 0～7 组成，例如 0321，表示八进制数 321。

十六进制整型常量以数字 0 和字母 X 即 0X 或 0x 开头，数码由 0～9，A～F（大小写均可）组成，例如 0Xb6、0x12，表示十六进制数 b6、12。

2.2.2 字符常量

字符常量是指用一对单引号括起来的一个字符。如 'a'，'b'，'A'，'-'，'!' 都是合法的字符常量。

字符常量中的单引号只起定界作用并不表示字符本身。单引号括起来的只能是单个字符，不能是字符串。

注意字符'5'和数字 5 的区别，前者是字符常量，后者是整型常量，它们的含义和在计算机中的存储方式都截然不同，字符常量'5'所表示的常量需要对照 ASCII 代码表。下面会详细介绍它们的区别。

用单引号括起来的字符表示字符常量，即转义字符。转义字符以反斜杠“\”开头，后面跟一个或几个字符，转义字符有具体的特定含义，与原字符意义不同，故称之为转义字符。例如，之前例题输出函数 printf()的格式控制部分用到的“\n”就是一个转义字符，其意思是“回车换行”。转义字符所对应的整型常量也同样需对照 ASCII 代码表。

也就是说，实际上这些字符不是以原始形态进行存放的，而是每个字符在计算机中对应着一个 ASCII 表值。常见的转义字符表见表 2-3。

表 2-3　常见的转义字符表

转义字符	含　义	ASCII 代码
\n	回车换行，将当前位置移到下一行开头	10
\t	横向跳格（跳到下一个输出区，一个输出区占 8 列）	9
\b	退格，将当前位置移到前一列	8
\r	回车，将当前位置移到本行开头	13
\f	换页，将当前位置移到下页开头	12
\\	反斜杠符“\”	92
\'	单引号符	39
\"	双引号符	34
\a	响铃	7
\ddd	1～3 位八进制数所代表的字符	
\xhh	1～2 位十六进制数所代表的字符	

【例 2.3】输出 a，b，c 的值。

```
void main()
{
    int  a, b, c;
    a=1;
    b=2;
    c=3;
    printf("%d\n\t%d␣␣%d\n␣␣%d␣␣%d\t\b%d\n",a,b,c,a,b,c);
/*要求按照给出格式输出 a，b，c 的值*/
}
```

程序运行结果如图 2-5 所示。

图 2-5　例 2.4 运行结果

分析：其中“␣”代表空格，程序在第一列输出 a 值 1，之后是“\n”换行，接着是“\t”，跳到下一制表位，再输出 b 值 2。空两个再输出 c 值 3，然后是“\n”换行，再空两格输出 a 的值 1，空两格再输出 b 的值 2。再遇到“\t”跳到下一制表位置，但下一转义字符“\b”又退回一格，所以 2 接着输出 3。

2.2.3　实型常量

在 C 语言中，实型常量一般都作为双精度来处理，并且只用十进制数表示。实型常量有两种书写格式：小数形式和指数形式。

（1）小数形式：它由符号、整数部分、小数点及小数部分组成。

例如：12.34，0.15，0.123，.123，123.，-12.0，-0.15，0.0，0. 都是合法的小数形式实型常量。

注意：其中的小数点是不可缺少的。例如 123.不能写成 123，因为 123 是整型常量，而 123. 是实型常量。

（2）指数形式：由十进制小数 e 指数或十进制小数 E 指数组成。

生活中，我们经常遇到数值很大的数据；数学中，我们常用科学计数法来表示这样的数；在 C 语言中，我们用类似于科学计数法的指数形式进行表示。由十进制数，加阶码标志“e”或“E”和阶码组成。其中，阶码只能为整数，可以带符号，并且 e（E）前必须有数字。

以下是合法的实数表示形式：

2.5E+5 或 2.5E5= 2.5×10^{5}，-2.5e-5= -2.5×10^{-5}

以下是非法的实数表示形式：

E7　　阶码标志 E 前没有数字

23.-E6　　负号位置不对

5.9e　　无阶码

2.5E-1.2　　阶码为小数

编写程序过程中，需要注意指数的表示形式，阶码必须为整数，可以带正号或负号，并且 e（E）前必须有数字。

2.2.4 字符串常量

字符串常量是由一对双引号括起来的字符序列。例如：“CHINA”，“very good”，都是合法的字符串常量。可以对一个字符串进行输出。例如：printf("Welcome to Beijing!");

要注意字符常量与字符串常量的区分。例如：'a'和"a"，前者是字符常量，后者是字符串常量，在编写程序时应注意使用要求。同时，它们所占内存空间不同。

'a'是字符变量内存只分配一个存储空间，只占一个字节。而"a"是字符串常量，内存对它分配两个存储空间包括 a 及'\0'，占两个字节。

在 C 中规定以字符'\0'作为字符串的结束标志。'\0'是 ASCII 代码为 0 的字符，即“空操作字符”也就是不起任何控制作用，也不是一个可以显示的字符。例如一个字符串“CHINA”，实际上在内存中的存储形式是：

C	H	I	N	A	\0

它的长度是 6 个字符，最后一个字符为'\0'。但在输出时不输出'\0'。系统在执行 printf("CHINA"); 时，一个一个地输出字符，直到遇到最后的'\0'字符，便知道字符串结束，停止输出。

字符串常量和字符常量之间有以下主要区别。

- 字符常量用单引号括起来单个字符，字符串常量用双引号括起来一个或多个字符。
- 可以把一个字符常量赋予一个字符变量，但不能把一个字符串常量赋予一个字符变量。
- 字符常量占有一个字节的内存空间，字符串常量占有的内存字节数为字符串中字符数加 1。
- 如果字符串常量中出现双引号，则要用反斜线\""将其转义，取消原有边界符的功能，使之仅作为双引号字符起作用。

  例如，要输出字符串：

```
He says:"How do you do."
```

应写成如下形式：

```
printf ("He says:\"How do you do.\"");
```

2.2.5　符号常量

有时为了使程序更加清晰以及便于修改，或者数值过长，影响代码长度，就可以用一个标识符来代表常量，也就是给某个常量取一个有意义的名字，这种常量称为符号常量。

符号常量在使用前必须先定义，定义的形式是：

#define　符号常量名　常量

这里用#define 指令，指定一个符号名称代表一个常量。#define 是 C 语言的预处理命令，它表示经定义的符号常量在程序运行前将由其对应的常量替换。

【例 2.4】宏定义。

```
#define     PI      3.1415926
#define     TRUE    1
#definr     FALSE   0
#define     STAR    '*'
```

分析：这里定义 PI、TRUE、FLASE、STAR 为符号常量，其值分别为 3.1415926，1，0，'*'。例如第一条指令，经过这样的指定后，此后程序中出现的 PI 都代表数值 3.1415926，它可以和常量一样进行运算，实际上在编译之后，符号常量就全部变成字面常量 3.1415926。这种用一个标识符代表一个常量的符号，称为符号常量。需要注意的是，符号常量也是常量，它的值在其作用域范围内不能再改变或赋值。如果出现下列语句给 PI 赋值

```
PI=3.1415926;
```

就为错误的形式。

使用符号常量对程序的编写及运用有如下优点。

“见名知义”，在定义符号常量名时应考虑增强程序的可读性，例如：

#define　PI　3.14 很容易让读者明白其中的 PI 代表圆周率。

“一改全改”，当程序中多处要用到同一个常量时，可以只改一处，其他都会全部改变，而不用一一去改。例如，在程序中要多次用到某商场的购物金额，如果金额用常数 100 来表示，当购物金额改变为 200 时，就需要在程序中的多处进行修改。若用符号常量 AMOUNT 来表示金额，则只修改一处即可。

定义符号常量的注意事项如下。

- 符号常量遵循标识符的命名规则。一般情况下，我们习惯将符号常量用大写字母来表示，变量名用小写表示，以示区分。
- 宏定义预处理命令，不是 C 语句，不必在末尾加分号（在 C 语言中由#开头都是命令）。

2.3　变　　量

变量是指在程序运行时其值可以改变的量。这里所说的变量与数学中的变量是完全不同的概念。在 C 语言以及其他各种常规程序设计语言中，变量是数据在内存中存储单元的名称。

2.3.1　变量的定义

在 C 语言中，变量定义的一般形式如下：

类型说明符 变量名表;

其中，类型说明符是 C 语言的一个有效的数据类型，如整型类型说明符 int，字符型类型说明符 char 等。变量表的形式是：变量名 1，变量名 2，…，变量名 n，即用逗号分隔的变量名的集合，最后用分号结束定义。例如：

```
int  a, b, c;        /*说明 a,b,c 为整型变量*/
char  ch;            /*说明 ch 为字符变量*/
double  d, e;        /*说明 d,e 为双精度实型变量*/
```

C 语言的语法格式较自由，把同类型多个变量的说明写在同一行是允许的。在 C 程序中，除了不能用关键字做变量名外，可以用任何合法的标识符做变量名。但是，一般提倡用能说明变量用途的、有意义的名字做变量命名，因为这样的名字对读程序的人有一定提示作用，有助于提高程序的可读性，尤其是当程序比较大、程序中的变量比较多时，这一点就显得尤其重要。

在 C 语言中，没有专门的字符串变量，如果需要将字符串存放在变量中，则需要用字符数组来存放，这将在后续章节中介绍。

程序里的一个变量可以看成一个存储数据的单元，它的功能就是存储数据。对变量的基本操作有两个：

（1）向变量中存入数据值，这个操作称为给变量“赋值”。

（2）取得变量当前值，以便在程序运行过程中使用，这个操作称为“取值”。

要对变量进行“赋值”和“取值”操作，程序里的每个变量都有一个变量名，程序是通过变量名来使用变量的。在 C 语言中，变量名作为变量的标识，其命名规则符合标识符的所有规定。

C 语言规定：程序里使用的每个变量都必须“先定义后使用”。也就是说，首先需要定义一个变量，然后才能够使用它。这样使用变量的优点是：

- 只有定义过的变量才可以在程序中使用，这使得变量名的拼写错误容易被发现。
- 定义的变量属于确定的类型，编译系统可方便地检查变量所进行运算的合法性。
- 在编译时，根据变量类型可以为变量确定存储空间，“先定义后使用”使程序效率高。

2.3.2 变量赋初值

在定义变量的同时给变量赋初值称为变量的初始化，以确保某些变量在程序运行时有确定的值。变量的初始化通常在变量定义时一起进行，通过赋值语句来实现。

在变量定义中赋初值的一般形式为：

类型说明符 变量 1=值 1，变量 2=值 2，…；

例如：
```
int i=3, j =10;        /*定义整型变量 i、j，并赋初值*/
double f=5.6597;       /*定义双精度浮点型变量 f，并赋初值*/
char b='A', c=100;     /*定义字符型变量 b、c，并赋初值*/
```

使用变量初始化的几点说明：

- 可以使被定义的变量的一部分赋初值。例如：int a，b，c=5；
- 不能对几个变量同时赋以一个值。

例如：int a=b=c=1；是错误的

应改写成：int a，b，c；a=b=c=1；

或 int a=1，b=1，c=1；

- 变量赋初值是在程序运行时完成的。

例如：int a=1；相当于 int a；a=1；两条语句。

注意：定义了变量但没赋值，变量中的数据则是一个随机数，这一点在程序设计中一定要注意。

2.4　运　算　符

C 语言的运算符应用范围很广泛，几乎所有的运算都可以由运算符实现。运算是对数据进行加工的过程，用来表示各种不同运算的符号称为运算符。除了一般高级语言具有的算术运算符、关系运算符、逻辑运算符以外，C 语言还提供了自增/自减运算符、赋值运算符、位运算符等。

按照运算功能分类，C 语言的运算符主要有以下几类：

（1）算术运算符　　+、-、*、/、%

（2）自增/自减运算符　　++、--

（3）赋值运算符　　=、+=、-=、*=、/=、%=

（4）关系运算符　　<、>、==、<=、>=、!=

（5）逻辑运算符　　!、&&、||

（6）位运算符　　<<、>>、~、|、^、&

（7）条件运算符　　？：

（8）逗号运算符　　，

（9）指针运算符　　*、&

（10）求字节数运算符　　sizeof

（11）强制类型转换运算符　　（类型）

（12）分量运算符　　、->

（13）下标运算符　　[]

（14）其他　　例如函数调用运算符()

当各种运算符混合时，需要根据运算符的优先级别进行运算。C 语言规定了各个运算符参与运算时的优先级别，当一个表达式中混合多种运算符时，计算是有先后次序的，这种计算过程中的先后次序称为相应运算符的优先级别。下面由高到低列出常用运算符的优先级别。

逻辑非运算符	!	高
算术运算符	+、-、*、/、%	↓
关系运算符	<、>、==、<=、>=、!=	
逻辑与	&&	
逻辑或	||	
赋值运算符	=、+=、-=、*=、/=、%=	低

C 语言的运算符按其在表达式中与运算对象的关系（连接运算对象的个数）可以分为以下三类。

（1）单目运算：一个运算符连接一个运算对象。

（2）双目运算：一个运算符连接两个运算对象。

（3）三目运算：一个运算符连接三个运算对象。

本章只介绍算术运算符、赋值运算符、逗号运算符、条件运算符等，其他运算符将在后续章节进行介绍。

2.4.1　算术运算符

算术运算符是在日常生活中最常用的。下面列出算术运算符的种类和功能。

1．基本算术运算符

+ 加法或正值运算符。例如，a+b 表示求 a 与 b 的和，+5 表示正值 5。
- 减法或负值运算符。例如，a-b 表示求 a 与 b 的差，-5 表示负值 5。
* 乘法。例如，a*b 表示求 a 与 b 的积。
/ 除法。例如，a/b 表示求 a 与 b 的商。
% 取余。例如，a%b 表示求 a 除以 b 的余数。

使用算术运算符时应注意以下几点：

- 减法运算符“-”不仅可以做减法，也可以作为取负值运算符，这时它为单目运算符。例如，-6、-(a+b)等。
- 除法运算符“/”如果遇到参与运算的变量均为整数，在多数编译系统中，其结果采取“向零取整”法，即去掉小数部分。例如，9/4=2，-9/2=–2。如果参与运算的两个数据有一个为实型，则按除法计算。例如，9/4.0=2.25。
- 求余运算符“%”，也称为“模运算符”，要求参与运算的两个操作数均为整型。其运算结果为两数相除所得的余数。例如，7%3=1，8%4=0，-9%5=-4。

【例 2.5】分析程序运行结果。

```
void main( )
{
    printf("d,%d\n",10/3,-10/3);
    printf("%f,%f\n",10/3.0,-10/3.0);
}
```

运行结果如图 2-6 所示。

```
3,-3
3.333333,-3.333333
```

图 2-6 例 2.5 运行结果

分析：本例中，10/3，-10/3 的结果均为整型，小数部分舍去。而 10.0/3 和-10.0/3 由于有实数参与运算，因此结果也为实型。

2．自增/自减运算符

自增/自减运算符均为单目运算符，即仅对一个运算对象进行运算，并且运算结果仍然赋给该运算对象。

自增/自减运算符包括两类：前置运算（++i,--i）和后置运算（i++,i--）。

前置运算：++i, --i 先使 i 的值加 1 或减 1，后使用 i。

后置运算：i++, i-- 先使用 i，后使 i 的值加 1 或减 1。

对于一个变量 i 进行前置运算或后置运算的结果是一样的，如++i 与 i++的结果是一样的，都执行 i=i+1。但++i 与 i++的不同之处在于++i 是先执行 i=i+1，再使用 i 的值；而 i++是先使用 i 的值后，再执行 i=i+1。

例如，i 的初值为 3，则执行下列每条赋值语句后，i 和 j 的值将产生变化。

j=++i; 先把 i 的值变为 4，再赋值给 j，j 的值为 4

j=i++; 先把 i 的值 3 赋给 j，j 的值为 3，后使 i 变为 4

j=--i; 先把 i 的值变为 2，再赋值给 j，j 的值为 2

j=i--; 先把 i 的值 3 赋给 j，j 的值为 3，后使 i 变为 2

例如，有 i=5，则执行以下输出语句后分析输出结果。

①printf("%d",++i);输出结果为“6”，执行后，i 的值为 6。

②printf("%d",i++);输出结果为“5”，执行后，i 的值为 6。

③printf("%d,%d",++i,++i);输出结果为“6, 7”，执行完后，i 的值为 7。

④printf("%d,%d",i++,i++);输出结果为“6, 5”，执行完后，i 的值为 7。

在多数系统中，对函数参数的求值顺序规定为“自右至左”，因此上两条输出语句才会有这样的输出结果。

【例 2.6】自增运算符的使用。

```
void main()
{
  int i, j, x, y;
  i=5, j=6;
  x=(i++)+(i++)+(i++);
  y=(++j)+(++j)+(++j);
     printf("%d, %d, %d, %d", i, j, x, y);
}
```

```
8,9,15,27
```

图 2-7　例 2.13 运行结果

运行结果如图 2-7 所示。

分析：在程序中，x 的结果为 15，y 的结果为 27。执行过程中，x 可以理解为三个 i 相加，先将 15 赋值给 x，后 i 自加三次，i 的结果值为 8；y 的值则是 j 先执行三次自加后，j 的结果为 9，再进行结果累加，结果为 27。

【例 2.7】自增/自减运算符的使用，分析运行结果。

```
void main()
{
    int i=5;
    printf("%d\n", ++i);    /*输出结果 6，i 的值为 6*/
    printf("%d\n", --i);    /*输出结果 5，i 的值为 5*/
    printf("%d\n", -i++);   /*输出结果-5，i 的值为 6*/
    printf("%d\n", -i--);   /*输出结果-6，i 的值为 5*/
}
```

```
6
5
-5
-6
```

图 2-8　例 2.7 运行结果

运行结果如图 2-8 所示。

使用自增/自减运算符需要注意以下几点。

- 自增/自减运算符的优先级高于算术运输符的优先级，几乎是常用运算符中优先级别最高的。
- 自增/自减运算符只能用于变量，而不能用于常量和表达式。因为常量的值是无法改变的，而表达式的自增或自减是不能实现的，因为无变量可存放。例如：5++或(a+b)++都是不合法的表达式。
- 自增/自减运算符的结合性是“自右至左”结合的。例如：有-i++，因结合方向为“自右至左”，所以它相当于-(i++)。例如：k=i+++j;应该理解为 k=(i++)+j 的形式，还是理解为 k=i+(++j)的形式呢？C 编译系统在处理时，尽可能多地自左至右将若干字符组成一个运算符，则应该是 k=(i++)+j 的形式。

3．算术表达式

运算符和运算对象按一定的规则结合在一起就构成了表达式。这里用算术运算符和括号将操作数连接起来符合 C 语言语法规则的式子即算术表达式。其运算对象可以是常量、变量、函数等。

例如：x+y，(a*b)/c-5+'a'，sin(x)+sin(y)等。

在 C 语言中，算术表达式的书写形式与普通数学表达式不同。在使用 C 语言的算术表达式时应注意以下几点。

- 乘号不能省。例如：xy 应写成 x*y。
- 所有符号写成一行。例如：$\frac{a+b}{c+d}$ 应改写成 C 语言表达式，(a+b)/(c+d)。
- C 语言中只能使用圆括号，可以嵌套使用，但左、右括号必须匹配。
- 上下角标不能直接写，需要转换。例如：b^2-4ac 应该写成 C 语言表达式，b*b-4*a*c。
- 避免两个运算符并置。例如：x*y/-z 应该为 x*y/(-z)。

调用标准数学函数时，自变量应写在一对括号内。例如：|–123|应改写成 fabs(-123)，sin30 应改写成 sin(30)。例如：cosα + sinβ 改写成 C 语句，应为 cos(alf)+sin(bate)。

- 三角函数的自变量应使用弧度。例如：sin50°应改写成 sin(50*3.1415926/180)。
- 要使所写的算术表达式与数学式子等价。

例如：$2x^2+3xy\sin 50°-|x-y|e^{2.3}$ 应改写成 C 语言表达式：

2*x*x+3*x*y*sin(50*3.1415926/180)-fabs(x-y)*exp(2.3)

例如：下列数学表达式

$$\frac{6\sin(x+y^2)}{a-\sqrt{x+y+1}}$$

转换成的 C 语言表达式为：

6*sin(x+y*y)/(a-sqrt(x+y+1))

其中，sin()和 sqrt()都是 C 语言提供的标准库函数中的成员。

2.4.2 赋值运算符

1．赋值运算符

赋值运算符用“=”表示，其功能是将一个数据赋给一个变量。例如：

```
n=12;
```

注意：赋值运算符“=”与数学中的等号完全不同，数学中的等号表示在该等号两边的值是相等的，而赋值运算符“=”是将其右边的数据存放到左边指定的内存变量中。

2．赋值表达式

由赋值运算符将一个变量和一个表达式连接起来的式子称为赋值表达式。它的一般形式为：

<变量名> <赋值运算符> <表达式>

赋值表达式的求解过程：计算赋值运算符右边“表达式”的值，并将计算结果赋值给赋值运算符左边的“变量”。赋值表达式的值就是赋值运算符左边“变量”的值。

例如：a=7;变量 a 的值为 7，这个赋值表达式“a=7”的值是 7。

上述一般形式中的表达式也可以是一个赋值表达式，所以可以有下面的式子：

```
a=(b=7);
```

即将 7 赋予变量 b，表达式 b=7 的值为 7，再赋予变量 a。根据赋值运算符的结合性（从右向左），上面的表达式等价于 a=b=7。下面是赋值表达式的例子：

```
a=b=c=8;
a=6+(b=3);
a=(b=3)*(c=4);
```

3．复合赋值运算符

在赋值运算符“=”之前加上其他运算符可以构成复合运算符，用于完成赋值复合运算操作。C 语言中的复合赋值运算符有：

+=、-=、*=、/=、%=、<<=、>>=、|=、&=、^=

由复合赋值运算符构成表达式的一般形式为：

<变量名>　<复合赋值运算符>　<表达式>

该式子等价于：

<变量名> = <变量名>　<运算符>　<表达式>

即先将变量和表达式进行指定的复合运算，然后将运算的结果值赋给变量。

例如：

a*=2 等价于 a=a*2

a/=b+5 等价于 a=a/(b+5)

注意：“a*=b+5”与“a=a*b+5”是不等价的，它等价于“a=a*(b+5)”，这里括号是必需的。

a-=1 等价于 a=a-1。

a%=3 等价于 a=a%3。

若 a=12，则 a+=a-=a*a；结果为 a=-264，该式等价于 a=a+(a=a-(a*a))。

若 a=2，则 a+=a*=a-=a*=3；结果为 a=0，该式等价于 a=a+(a=a*(a=a-(a=a*3)))。

说明：

- 赋值运算符的优先级最低。
- 赋值运算符的结合方向为：自右向左。
- 赋值运算符左侧必须是变量，不能是常量或表达式。

2.4.3　逗号运算符

1．逗号运算符

逗号运算符“,”是一种特别的运算符，它不是指那些在同时定义的几个变量或函数调用的几个参数之间起分隔作用的逗号，而是用逗号将两个或多个表达式连接起来，形成逗号表达式。

2．逗号表达式

一般形式：表达式 1，表达式 2，…，表达式 n

求解过程为自左至右，先求解表达式 1，再求解表达式 2，…，求解表达式 n，最后求解的表达式的值即整个逗号表达式的值。

例如：2*5，6+9

先计算前一个表达式 2*5 得到值 10，再计算后一个表达式 6+9，得到值 15，整个逗号表达式的结果为后一个表达式的值 15。

例如：

```
a=3*5,a*4              /*a=15，表达式值 60*/
a=3*5,a*4,a+5          /*a=15，表达式值 20*/
x=(a=3,6*3)            /*赋值表达式，表达式值 18，x=18*/
x=a=3,6*a              /*逗号表达式，表达式值 18，x=3*/
a=1;b=2;c=3;
```

```
printf("%d,%d,%d",a,b,c);          /*1,2,3*/
printf("%d,%d,%d",(a,b,c),b,c);    /*3,2,3*/
```

使用逗号运算符需要注意以下几点。

- 一个逗号表达式可以与另一个表达式组成新的逗号表达式，例如：(a=3*4, a*2)，a+5
- 逗号运算符是所有运算符中优先级别最低的一个运算符。例如：

```
x=y=3, y+4      逗号表达式结果为 7，x 值为 3
x=(y=3, y+4)    逗号表达式结果为 7，x 值为 7
```

第一个表达式为逗号表达式，按顺序计算，x 与 y 的结果均为 3，逗号表达式的结果值为 7；第二个表达式为赋值表达式，是把逗号表达式的结果值赋给 x，逗号表达式的结果值为 y+4 的值 7，因此 x 的结果值为 7。

- 使用逗号表达式的目的往往不在于要取得整个逗号表达式的值，而是要让逗号运算符两边的表达式都运算一遍。逗号表达式经常用于后面章节将要讲到的 for 循环中。例如：

```
for(i=0; i<10; i++, j++)
```

在 i++和 j++之间的就是一个逗号运算符，这个表达式作用是每循环一次，逗号运算符两边的变量 i 和 j 均增加 1。

逗号并不是出现在任何地方都作为逗号运算符，有些情况只起间隔符的作用，不构成逗号表达式。例如：int a, b, c; 以及 printf("%d,%d",x,y); 这些逗号都属于间隔符。

【例 2.8】运行程序，分析运行结果。

```
void main()
{
   int a,b;
   a=(b=2，b+3)；
   printf("%d，%d", (a，b+a), b);
}
```

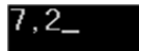


图 2-9 例 2.8 运行结果

运行结果如图 2-9 所示。

2.4.4 条件运算符

1．条件运算符及表达式

条件运算符是 C 语言中唯一的三目运算符，它用两个符号“?”和“:”把三个运算对象连接在一起。符号“?”前是一个关系表达式或由逻辑运算符连接起来的组合关系表达式，用于表示决定条件表达式取值的条件，中间和后面的两个表达式分别代表条件表达式可取的两个值。当条件运算结果为真时，取前一个的值，否则取后一个的值。

2．条件表达式

一般形式：表达式 1? 表达式 2: 表达式 3

先计算表达式 1 的值，若非零则执行表达式 2，把表达式 2 的值作为整个条件表达式的值；若表达式 1 的值为零，则执行表达式 3，并且把表达式 3 的值作为整个条件表达式的值。

例如：

x>y?6:7，表示当 x>y 成立时，条件表达式的值为 6，否则条件表达式的值为 7。

```
(a==b)? 'Y': 'N'
```

```
(x%2==1)?1:0
(x>=0)?x:-x
(c>= 'a' && c<= 'z')?c- 'a'+ 'A':c
```

条件运算符的结合性是自右向左结合，当多个条件表达式嵌套使用时，每个后续的“:”总与前面最近的、没有配对的“?”相联系。

例如，条件表达式“a>0 ? a/b:a<0 ? a+b:a-b”等价于 a>0 ? a/b: (a<0 ? a+b:a-b)。

使用条件表达式可以使程序简洁明了。例如，赋值语句“z=(a>b)?a:b”中使用了条件表达式，很简洁地表示了判断变量 a 与 b 值，将最大值赋给变量 z 的功能。所以，使用条件表达式可以简化程序。

【例 2.9】从键盘接收一个字符，要求只把输入的小写字母转换成大写字母，其他字符不变，并显示结果。

```
#include <stdio.h>
main()
{
   char ch,c;
   scanf("%c",&ch);
   c=ch>='a'&&ch<='z'?ch-32:ch;
   printf("%c\n",c);
 }
```

运行结果如图 2-10 所示。

图 2-10　例 2.9 运行结果

2.4.5　求字节长度运算符及其表达式

1．求字节长度运算符

和其他运算符不同的是，求字节运算符是由一个关键字 sizeof 表示的，它用于计算一种数据类型在计算机内存中所占用的字节数。不论是基本的数据类型还是复杂的构造类型，都可以用它来计算。

2．求字节长度运算表达式

一般形式：sizeof(表达式)

例如：sizeof(char)　　计算一个字符类型数据占用的字节数，计算结果为 1

sizeof(double)　　计算一个双精度浮点型数据的长度，计算结果为 8

sizeof (a*b)　　求运算结果在内存中所占用的字节数

sizeof(3*1.46/7.28)　　求运算结果在内存中所占用的字节数，为 8

【例 2.10】用 sizeof 测试 VC++ 6.0 中各种数据类型所占内存的字节数。

C 语言源程序如下：

```
#include <stdio.h>
void main()
{
    char ch='A';
    int x=7,y=8;
    float a=3.27f,b=6000.0f;
  printf("char:%d\n",sizeof(ch));
```

```
    printf("short int:%d int:%d long int:%d\n",sizeof(short int),
          sizeof(int),sizeof(long int));
    printf("float:%d\n",sizeof(a));
    printf("double:%d long double:%d\n",sizeof(double),sizeof(long double));
    printf("int express:%d\n",sizeof(x+y));
    printf("float express:%d\n",sizeof(a+b));
    printf("character express:%d\n",sizeof('a'-'0'));
}
```

运行结果如图 2-11 所示。

```
char:1
short int:2 int:2 long int:4
float:4
double:8 long double:10
int express:2
float express:8
character express:2
```

图 2-11　例 2.10 运行结果

2.4.6　位运算符

1．位运算符

在很多系统程序中，常要求在位（bit）一级进行运算或处理。C 语言提供了位运算的功能，这使得 C 语言也能像汇编语言一样用来编写系统程序。

C 语言提供了六种位运算符。

（1）&：按位与运算符。

（2）|：按位或运算符。

（3）^：按位异或运算符。

（4）~：取反运算符。

（5）<<：二进制左移运算符。

（6）>>：二进制右移运算符。

按位运算符的两个运算对象都只能是整型或者字符型。左移和右移运算符右边的数值表示移动的二进制位数，该数值不允许取负值，其值也不允许超过移位运算符左边数值所占用的实际位数。

2．位运算表达式

1）按位与运算符及其表达式

按位与运算符“&”是双目运算符。其功能是将参与运算的两个数对应的二进制位相与。只有对应的两个二进位均为 1 时，结果位才为 1，否则为 0，即 0&0=0，0&1=0，1&0=0，1&1=1。参与运算的两个数均以补码方式出现。

【例 2.11】计算 3&5。

3&5 的结果并不等于 8，计算过程应先把 3 和 5 写成补码形式，再按位与运算。可写算式如下：

00000011（3 的二进制补码）
& 00000101（5 的二进制补码）
00000001（1 的二进制补码）

即得 3&5=1。

按位与运算有以下特殊用途。

- 清零作用。按位与运算通常用来对某些位清零或者保留某些位。例如：把 a 的高 8 位清 0，保留低 8 位，可看成 a&255（255 的二进制数为 0000000011111111）。
- 保留特定位。例如：对于某个数据只想保留此数第 2、4、6 位的值。这时，可以与一个数进行&运算，此数的第 2、4、6 位为 1，其余各位为 0，即将该数按位与上二进制数 01010100。例如：

00101101（45 的二进制补码）
& 01010100（84 的二进制补码）
00000100（4 的二进制补码）

即得 45&84=4

【例 2.12】运行程序分析结果。

```
void main()
{
    int a=4,b=5,c;
    c=a&b;
    printf("a=%d, b=%d, c=%d\n",a,b,c);
}
```

运行结果如图 2-12 所示。

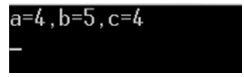


图 2-12　例 2.12 运行结果

2）按位或运算符及其表达式

按位或运算符“|”是双目运算符。其功能是将参与运算的两个数对应的二进制位相或。只要对应的两个二进位有一个为 1 时，结果位就为 1，即 0|0=0，0|1=1，1|0=1，1|1=1。参与运算的两个数均以补码出现。

【例 2.13】计算 4|5。

计算过程应先把 4 和 5 写成补码形式，再按位与运算。可写算式如下：

00101101（4 的二进制补码）
|01010100（5 的二进制补码）
01110101（117 的二进制补码）

即得 4|5=117。

【例 2.14】运行程序分析结果。

```
void main()
{
    int a=7,b=5,c;
    c=a|b;
    printf("a=%d, b=%d, c=%d\n",a,b,c);
}
```

运行结果如图 2-13 所示。

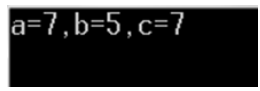


图 2-13　例 2.14 运行结果

3）按位异或运算符及其表达式

按位异或运算符“^”是双目运算符。其功能是将参与运算的两个数对应的二进制位相异或。

当两数对应的二进位不同时，其结果为 1，否则结果为 0，即 0^0=0，0^1=1，1^0=1，1^1=0。参与运算的两个数均以补码出现。

【例 2.15】计算 6^5。

计算过程应先把 6 和 5 写成补码形式，再按位与运算。可写算式如下：

```
  00000110（6 的二进制补码）
^ 00000101（5 的二进制补码）
----------------------------
  00000011（3 的二进制补码）
```

即得 6^5=3。

【例 2.16】运行程序分析结果。

```
void main()
{
    int a=10;
    a=a^15;
    printf("a=%d\n",a);
}
```

图 2-14　例 2.16 运行结果

运行结果如图 2-14 所示。

4）求反运算符及其表达式

求反运算符“~”为单目运算符，具有右结合性。其功能是将参与运算的数据的各二进制位按位求反。例如：~7=~(00000111)=11111000。由于计算机中存放补码形式，所以~7=–8。

5）左移运算符及其表达式

左移运算符“<<”是双目运算符。其功能是把左移运算符左边的数据按二进制位向左移动若干位，由左移运算符右边的数指定移动的位数，高位舍弃，低位补 0。其表达式为：x<<n。

例如：a<<2 指把 a 的各二进位向左移动 2 位。如 a=00000111（十进制 7），向左移 2 位后为 00011100（十进制 28）。

6）右移运算符及其表达式

右移运算符“>>”是双目运算符。其功能是把右移运算符左边的数据按二进制位向右移若干位，由右移运算符右边的数指定移动的位数。低位舍弃，无符号数位数左边高位补 0，有符号位数，左边高位不变。其表达式为 x>>n。

例如：a=7，a>>2 表示把 00000111 右移为 00000001（十进制 1）。

2.4.7　类型转换

在 C 语言中，整型、实型和字符型数据之间可以进行混合运算。不同类型的数据进行赋值、混合运算时，不同的数据类型要先转换成同一类型，然后再进行运算。

类型转换可归纳成两种转换方式：隐式类型转换和强制类型转换。

1．隐式类型转换

隐式类型转换也称自动类型转换，是由系统自动完成的，这种类型转换不会体现在 C 语言源程序中。但是，程序设计人员必须了解这种自动转换的规则及其结果，否则会引起对程序执行结果的误解。

隐式类型转换通常发生在运算、赋值、输出、函数调用中。函数调用转换——实参与形参类型不一致时转换，具体转换规则将在后面章节中讲述。

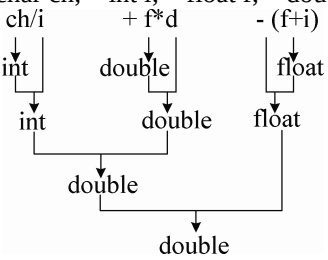
1）运算转换

C 语言允许进行整型、实型、字符型数据的混合运算，在运算时，会将不同类型的数据转换成同一类型后再进行运算。转换的规则如图 2-15 所示。

- 横向箭头表示运算时必定的转换，即运算时将表达式中所有 char 和 short 型数据转换成 int 型后再进行运算。
- 纵向的箭头表示当表达式中含有不同类型的数据时的转换方向，如 int 型和 double 类型的数据进行运算，先将 int 型转换成 double 类型后再进行运算，运算结果为 double 类型。

下面给出类型转换的示例，以加深理解。

例如 char ch;　int i;　float f;　double d;



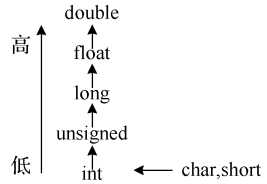


图 2-15　隐式类型的转换规则

以上步骤中的类型转换都是 C 语言编译系统自动完成的。计算机运算时从左向右扫描表达式，先扫描到 ch/i+，“/”比“+”优先，先算 ch/i，将 ch 转换成 int 型运算，结果为 int 型，再做运算符“+”和“*”的比较，“*”优先级较高，“+”先不运算，再向后扫描，遇到“-”，“-”优先级低于“*”，所以进行 f*d 的运算，将 f 转换成 double 进行运算，结果为 double，从左向右有“+”和“-”，“+”在前，先计算“+”，将“+”左边的 int 转换成 double 类型与其右边的 double 类型相加，结果为 double 类型，“-”和“(”相比，括号优先级高，先计算括号里面的，将 int 型转换成 float 类型，运算结果为 float 型，最后进行“-”运算，“-”号运算符左边为 double 类型，所以将右边的 float 类型转换成 double 类型，进行“-”运算，结果为 double 类型。

2）赋值转换

在进行赋值运算时，如果赋值运算符两边的数据类型不一致，将由系统进行自动类型转换，转换原则是以“=”左边的变量类型为准，先将赋值号右边表达式类型转换为左边变量的类型，然后赋值。

在赋值时要注意以下情况：

- 将实型数据（单、双精度）赋给整型变量，舍弃实数的小数部分。

例如：float a=6.7; int b; b=a; 则 b 的值为 6。

- 将整型数据赋给单、双精度实型变量，数值不变，但以浮点数形式存储到变量中。

例如：float a; int b=5; a=b; 则 a 的值为 5.000000。

- 将 double 型数据赋给 float 型变量时，截取其前面 7 位有效数字，存放到 float 变量的存储单元中（32bit）。但应注意数值范围不能溢出。将 float 型数据赋给 double 型变量时，数值不变，有效位数扩展到 16 位。
- 字符型数据赋给整型变量时，由于字符只占一个字节，而整型变量为 4 个字节，因此将字符数据（8bit）放到整型变量的低 8 位中。其他位数据有以下两种情况。

（1）如果所使用的系统将字符处理为无符号的量或对 unsigned char 型变量赋值，则将字符的 8 位放到整型变量的低 8 位，其余位补 0。

（2）当所使用的系统将字符处理为带符号的量（signed char）（如 VC++ 6.0）时，若字符最高

位为 0，则将字符数据（8 位）置于整型变量的低 8 位，整型变量其余位补 0；若字符最高位为 1，则整型变量其余位全补 1。这称为符号扩展，这样做的目的是使数值保持不变。

- 将一个 int，short，long 型数据赋给一个 char 型变量时，只是将其低 8 位原封不动地送到 char 型变量（即截断）。
- 不同类型的整型数据间的赋值按照存储单元的存储形式直接传送。由长整型数赋值给短型整数，截断直接传送；由短型整数赋值给长型整数，低位直接传送，高位根据被传送整数的符号进行符号扩展。

2．强制类型转换

在 C 语言里，一种类型的数据可以强制转换为另一种类型的数据。数据类型强制转换符是将类型符放在一对圆括号里，并把它们放在需要转换类型的数据前。例如：要把浮点变量 a 的值转换成整型，可以用表达式(int)a 来完成。又如表达式(char)(4.2+3.3)将结果转换成字符型量，小数点后面的部分被舍弃。要注意，表达式后面的括号不能省略，否则意思就不一样了。

一般形式：（类型说明符）表达式

例如：(int)x　　将 x 的强制转换为整型

(int)a+b;　　将 a 强制转换为整型后与 b 相加

(float)(7%4)　　将 7 对 4 的求余结果强制转换为单精度浮点型

(double)(x+y)　　将 x 与 y 的相加的和强制转换为双精度浮点型

使用强制类型转换运算符应注意以下几点：

- 类型说明符和表达式都必须加上圆括号，单个变量时可以不加括号，例如：把(int)(a+b)写成(int)a+b 就只转换了变量 a 的类型，而没有转换变量 b 及整个结果的类型。
- 无论是强制转换还是自动转换，都是为了本次运算的需要而对变量的数据的类型进行临时转换，并不改变数据说明时对此变量定义的类型。

【例 2.17】运行程序，分析运行结果。

```
void main()
{
int a;
    float b=5.7, c;
    a=(int)b;
    c=(float)a+b;
    printf("a=(int)b=%d, c=(float)a+b =%f",a,c);
}
```

运行结果如图 2-16 所示。

```
a=(int)b=5,c(float)a+b=10.700000
```

图 2-16　例 2.17 运行结果

本 章 小 结

（1）熟悉常量、变量和标识符的命名规则，熟悉常用的关键字。

（2）C 语言中基本的数据类型，包括整型（int）、单精度浮点型（float）、双精度浮点型

（double）、字符型（char），整型数据前可加限定词（short、long、signed、unsigned），有时可省略 int 本身。

（3）当多种不同类型数据混合在一个表达式中时，应注意数据的自动转换原则，以确保得出正确结果。

（4）C 语言中提供了丰富的运算符，包括算术运算符、自增/自减运算符、赋值运算符、关系运算符、逻辑运算符、位运算符、条件运算符、逗号运算符、指针运算符、求字节数运算符、强制类型转换运算符、分量运算符等。本章主要介绍其中的常用运算符和表达式，有些运算符（如关系运算符）将在后面章节中陆续介绍。一般而言，单目运算符优先级较高，赋值运算符优先级低。算术运算符优先级较高，关系和逻辑运算符优先级较低。多数运算符具有左结合性：单目运算符、三目运算符、赋值。

（5）表达式是由运算符连接常量、变量、函数所组成的式子。每个表达式都有一个值和类型。表达式求值按运算符的优先级和结合性所规定的顺序进行。

应重点掌握这些运算符的作用，稍有疏忽就会出错，初学者应在计算机上进行实验，以验证理解是否正确。

习　题　2

一、单选题

1．整型（int）数据在内存中的存储形式是（　　）。

A．原码　　B．补码　　C．反码　　D．ASCII 码

2．C 语言中，整数-8 在内存中的存储（设 2 个字节）形式是（　　）。

A．1111 1111 1111 1000　　B．1000 0000 0000 1000

C．0000 0000 0000 1000　　D．1111 1111 1111 0111

3．下列常数中不能作为 C 的常量的是（　　）。

A．0xA6　　B．4.5e-2　　C．3e2　　D．0584

4．下列可以正确表示字符型常数的是（　　）。

A．"c"　　B．'\t'　　C．"\n"　　D．297

5．C 程序中，运算对象必须为整型数据的运算符是（　　）。

A．++　　B．%　　C．/　　D．&

6．下面的变量说明中，（　　）是正确的。

A．char:a, b, c;　　B．char a; b; c;　　C．char a, b, c;　　D．char a, b, c

7．C 语言中，下列属于构造类型的是（　　）。

A．整型　　B．字符型　　C．实型　　D．数组类型

8．C 语言中，下列属于基本类型的是（　　）。

A．整型、实型、空类型　　B．结构体类型、字符型

C．空类型、枚举类型　　D．整型、实型、字符型

9．C 语言中，不能用来表示整型常数进制是（　　）。

A．八进制　　B．十进制　　C．十六进制　　D．二进制

10．下列字符变量表示错误的是（　　）。

A．a=65　　B．a="b"　　C．a='c'　　D．a='a'

11．C 语言中，合法的数值常量是（　）。

A．5H　　B．5　　C．17E　　D．081

12．C 语言中，“define　PRICE　1.5”将 PRICE 定义为（　）。

A．字符常量　　B．符号常量　　C．实型常量　　D．变量

二、填空题

1．表达式（double）(1/3+.5*3+5%3）的计算结果为_____。

2．表达式（int）(1/3+.5*3+5%3）的计算结果为_____。

3．有语句：“int x=1, y=1;”，则执行逗号表达式 y=3, x++, x+5 后，该表达式的值是_____，变量 x 的值是_____，变量 y 的值是_____。

4．C 语言中，程序运行期间，其值不能被改变的量称为_____。

5．在 C 语言程序中，用关键字_____定义基本整型量，用关键字_____定义单精度实型变量，用关键字_____定义双精度实型变量。

6．若 w=1, x=2, y=3, z=4，则条件表达式 x<x？w:y<z？w:y 的结果为_____。

7．C 语言中，标识符只能由_____、_____和_____三种字符组成，且第一个字符必须为_____或_____。

8．C 语言中，要求对所有用到的变量，遵循先_____后_____的原则。

9．C 语言中，实数有两种表现形式，即_____和_____形式。

10．C 语言中规定，在变量定义的同时也可以给变量赋初值，称为_____。

11．若有以下类型说明语句：char w；int x；float y；double z;，则表达式 w*x+z-y 的结果为_____类型。

三、读程序写结果

1．写出以下程序的运行结果。

```
main()
{   int x,y,z;
    x=1;
    y=5;
    z=x%y;
    z++;
    printf("%d\n",z);
}
```

2．写出以下程序的运行结果。

```
main()
{   float x;
    x=1.5;
    x=++;
    printf("%f\n",x);
}
```

3．写出以下程序的运行结果。

```
main()
{   int x;
    x=10;
```

```
    printf("%d\n",(--x*3/5));
}
```

4. 写出以下程序的运行结果。

```
main()
{   int x;
    x=10;
    printf("%d\n",(x--*3/5));
}
```

5. 写出以下程序的运行结果。

```
main()
{   int x;
    float y;
    x=10/3; y=10%3;
    printf("%d,%f\n",x,y);
}
```

6. 写出程序的运行结果。

```
void main()
{   int i,j,k,m;
    i=4;j=5;
    k=++i+j++;
    m=++j+i;
    printf("%d,%d,%d,%d",i,j,k,m);
}
```

7. 写出程序的运行结果。

```
void main()
{  float x,y;
   int a,b;
   x=3.6;y=2.5;
   a=(int)x;
   b=(a+3)/y;
   printf("%d,%d", a, b);
}
```

8. 写出程序的运行结果。

```
main()
{   char c1='A'; int i=10;
    c=c+10;
    i=c%i;
    printf("%c，%d\n", c,i);
}
```

9. 已知 A 的 ASCII 码为 65，写出程序的运行结果。

```
#include<stdio.h>
main()
{   char c1='A',c2='Y';
```

```
    printf("%d,%d\n",c1,c2);
}
```

10. 写出程序的运行结果。

```
void main()
{
    int i,j;
    i=16;
    j=(i++)+i;
    printf("%d\n", j);
    i=15;
    printf("%d,%d", ++i, i);
}
```

11. 写出程序的运行结果。

```
#include<stdio.h>
#define sum 10+20
main()
{   int b=0,c=0;
    b=5;
    c=sum*b;
    printf("%d\n",c);
}
```

12. 写出程序的运行结果。

```
#include <stdio.h>
void  main()
{
    float  x;
    int  i;
    x=3.6;
    i=(int)x;
    printf("x=%f,i=%d",x,i);
}
```

第 3 章　顺序结构程序设计

3.1　算　　法

通常，一个程序包括算法、数据结构、程序设计方法、语言工具和环境这 4 个方面。其中，算法是核心，算法是解决问题的工具和步骤。

做任何事情都有一定的步骤，编写一个程序时也不例外，也要先设计解决方法。一个程序主要包括两部分：一是对数据的描述，二是设计解决步骤。这就是程序的算法。

3.1.1　算法的概念

算法与程序设计以及数据结构密切相关，是解决一个问题的完整的步骤描述，是解决问题的策略、规则、方法。算法的描述形式有很多种，如传统流程图、结构化流程图计算机程序语言等。下面介绍算法的一些相关内容。

3.1.2　算法的特性

一个算法是为解决某一特定类型的问题而制定的一个实现过程，它具有下列特性。

1．有穷性

一个算法必须在执行有穷步之后结束且每一步都可在有穷时间内完成，不能无限地执行下去。例如，要编写一个由小到大的整数累加的程序，这时候要注意必须设一个整数的最上限，也就是加到哪个数为止。若没有这个最上限，那么程序将无终止地运行下去，也就是常说的死循环。

2．确定性

算法的每一个步骤都应当是确切定义的，对于每一个过程不能有二义性，对要执行的每个动作必须做出严格而清楚的规定。

3．可行性

算法中的每一步都应当能有效地运行，也就是说，算法应是可行的要求最终得到正确的结果。例如下面一段程序：

```
int x,y,z;
scanf ("%d,%d,%d",&x,&y,&z);
if(y==0)
z=x/y;
```

在这段代码中，z=x/y;就是一个无效的语句，因为 0 是不可以做分母的。

4．输入

一个算法应有零个或多个输入，输入是在执行算法时需要从外界取得必要的如算法所需的初始量等一些信息。例如：

```
int a,b,c;
scanf ("%d,%d,%d",&a,&b,&c);
```

上面代码就是有多个输入，又如：

```
main()
{
    printf("hello world! ");
}
```

上面代码需要零个输入。

5．输出

一个算法有一个或多个输出。什么是输出？输出就是算法最终所求的结果。编写程序的目的就是要得到一个结果。如果一个程序运行下来没有任何结果，那么这个程序本身也就失去了意义。

3.1.3 算法的优劣

衡量一个算法的好坏，通常要从以下几个方面来分析。

1．正确性

正确性指所写的算法能满足具体问题的要求，即对任何合法的输入算法都会得出正确的结果。

2．可读性

可读性指算法被写好之后，该算法被理解的难易程度。一个算法可读性的好坏十分重要。如果一个算法比较抽象，难于理解，那么这个算法就不易交流和推广使用，对于修改、扩展、维护都十分不方便。因此，在写一个算法的时候，要尽量将该算法写得简明易懂。

3．健壮性

一个程序完成后，运行该程序的用户对程序的理解各有不同，并不能保证每一个人都能按照要求进行输入。健壮性就是指当输入的数据非法时，算法也会做出相应的判断，而不会因为输入的错误造成瘫痪。

4．时间复杂度与空间复杂度

时间复杂度简单地说就是算法运行所需要的时间。不同的算法具有不同的时间复杂度。当一个程序较小时，感觉不到时间复杂度的重要性；当一个程序特别大时，便会察觉到时间复杂度实际上是十分重要的。因此，写出更高速的算法一直是算法不断改进的目标。空间复杂度是指算法运行所需的存储空间的多少，随着计算机硬件的发展，空间复杂度已经不再显得那么重要。

3.1.4 算法的描述

算法包含算法设计和算法分析两方面内容。算法设计主要研究怎样针对某一特定类型的问题设计出求解步骤，算法分析则要讨论所设计出来的算法步骤的正确性和复杂性。对于一些问题的求解步骤，需要一种表达方式，即算法描述。他人可以通过这些算法描述来了解算法设计者的思路。表示一个算法，可以用不同的方法，常用的有自然语言、流程图、N-S 流程图等。下面对算法的描述做进一步的介绍。

在写一个程序之前，需要先为其设计算法。那么，怎样来表示这个算法呢？其实，算法表示方法有多种，常用的有以下几种。

1．用自然语言表示算法

自然语言就是人们常用的语言，这种表示方式通俗易懂。简单地说，算法就是为解决一个问题而采取的方法和步骤。这里要研究的是能在计算机执行的算法，即计算机算法。下面举一个简单例子，以了解什么是算法。

【例 3.1】小明欲乘公交车从 A 点到 C 点，有甲、乙两条路可以选择，不同的是，如果选择乙路，需要在 D 点转车，在 D 点等车需要 3 分钟，甲路线不需转车，如图 3-1 所示。问：走哪条路到达 C 点所需时间最少？假设各线路公交车的行驶速度相同，图中数字表示两点间距离，单位为公里。

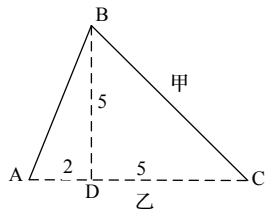


图 3-1　小明乘车路线图

算法设计如下。

步骤 1：计算 AB 距离，计算 AC 距离。

步骤 2：假设公交车速度为 v，计算走甲路所需的时间 T1。

步骤 3：计算走乙路所需的时间 T2。

步骤 4：比较 T1、T2，判断出走哪条路。

其实，解决同一问题时，方法有多种，这些算法也有“好”、“坏”之分。在以后的学习中，读者可以逐步总结较好的算法，这也是计算机学习中的一个重要课程。

这种表示方法多用于很简单的问题，如例 3.1 所示。

【例 3.2】求 n!。

①定义 3 个变量 i、n 及 mu1，并为 i 和 mul 均赋初值 1。

②从键盘中输入一个数赋给 n。

③将 mul 乘以 i 的结果赋给 mul。

④i 的值加 1，判断 i 的值是否大于 n，如果大于 n，则执行步骤⑤，否则执行步骤③。

⑤将 mul 的结果输出。

【例 3.3】任意输入 3 个数，求这 3 个数中的最小数。

①定义 4 个变量，分别为 x、y、z、min。

②输入大小不同的 3 个数分别赋给 x、y、z。

③判断 x 是否小于 y，如果小于，则将 x 的值赋给 min，否则将 y 的值赋给 min。

④判断 min 是否小于 z，如果小于，则执行步骤⑤，否则将 z 的值赋给 min。

⑤将 min 的值输出。

以上介绍的实例的算法实现过程就是采用自然语言来描述的。从上面的描述中会发现采用自然语言描述的好处就是易懂。但是，采用自然语言进行描述也有很大的弊端，就是容易产生歧义，例如将例 3.2 步骤③中的“将 mul 乘以 i 的结果赋给 mul”改为“mul 等于 i 乘以 mul”，这样就产生了歧义，并且用自然语言来描述较为复杂的算法就显得不是很方便，因此一般情况下不采用自然语言来描述。

2．用流程图表示算法

流程图是一种传统的算法表示法，它用一些图框来代表各种不同性质的操作，用流程线来指示算法的执行方向。由于它直观形象，易于理解，所以应用广泛，特别是在语言发展的早期阶段，只有通过流程图才能简明地表述算法。

1）流程图的基本表示图形

流程图是使用一些图框来表示各种操作的。起止框是用来标识算法开始和结束的；判断框的

作用是对一个给定的条件进行判断，根据给定的条件是否成立来决定如何执行后续的操作；连接点是将画在不同地方的流程线连接起来。

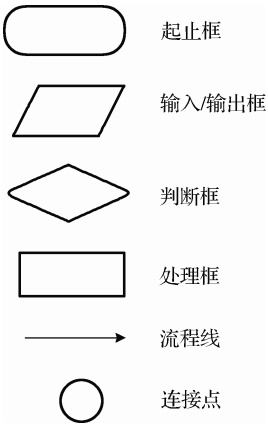


图 3-2 流程图的基本表示图形

下面通过一个例子来介绍这些图框如何使用。图 3-2 列出了流程图中的各种基本表示图形，在以后的学习中会经常看到这种算法表示方式，所以一定要学会读懂程序流程图。

2）三种基本结构

C 语言是一种结构化程序设计语言，摒弃了早期程序设计语言的一些弊端，如多入口多出口等，任何 C 程序都可以用三种基本结构来表示。Bohra 和 Lacopini 为了提高算法的质量，经研究提出了三种基本结构，即顺序结构、选择结构和循环结构，因为任何一个算法都可由这三种基本结构组成。这三种基本结构之间可以并列，可以相互包含，但不允许交叉，不允许从一个结构直接转到另一个结构的内部。

因为整个算法都是由三种基本结构组成的，所以只要确定好三种基本结构的流程图的画法，就可以画出任何算法的流程图，即顺序结构、选择结构、循环结构。下面介绍这三种基本结构用流程图是如何表示的。

（1）顺序结构

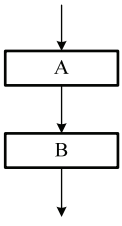
顺序结构表达的是，在程序执行时，按照从上到下的次序依次完成各处理，如图 3-3 所示，表示先执行处理 A，再执行处理 B。

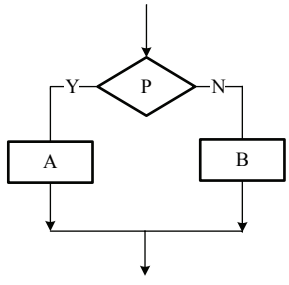
（2）选择结构

选择结构又称为分支结构，如图 3-4 所示。该结构表示程序执行到这里时，先对表达式 P 做一个判断，表达式为真时，执行处理 P；表达式不成立时，执行处理 B。

（3）循环结构

循环结构适合于在满足一定条件下，重复执行某些操作的情形，如图 3-5 所示，表示当满足条件 P 时，执行处理 A，然后再重复判断表达式 A，当不能满足条件 P 时，退出该循环，继续向下执行。





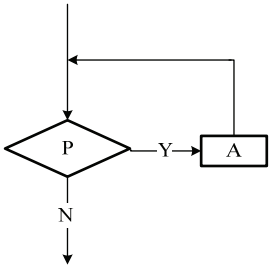


图 3-3 流程图的顺序结构表示　图 3-4 流程图的选择结构表示　图 3-5 流程图的循环结构表示

例 3.1 中的算法用流程图表示，如图 3-6 所示。

3．用 N-S 图表示算法

这里我们再简单地介绍怎样用盒图（又称为 N-S 图）来表示算法。C 语言的三种结构对应用 N-S 图的表示方式如下。

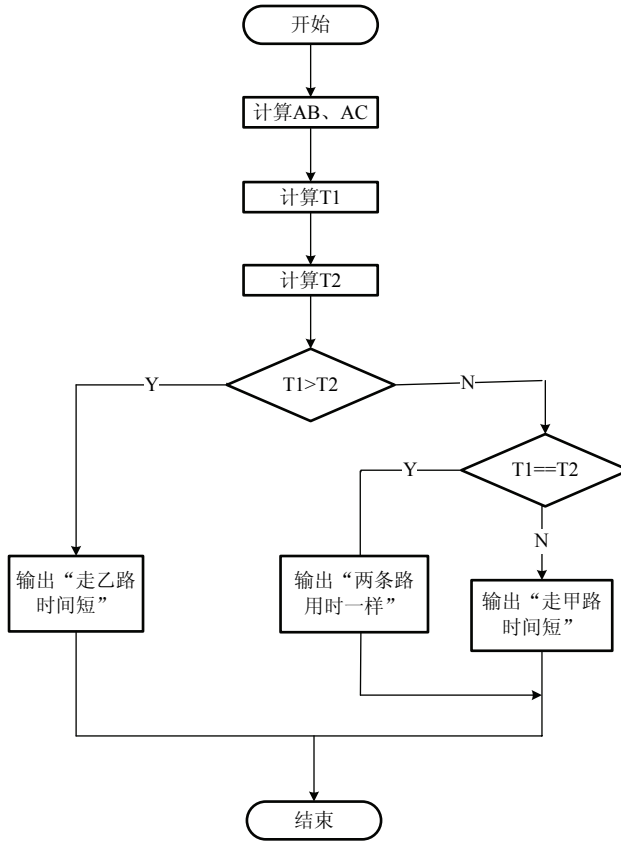


图 3-6　例 3.1 的程序流程图

1）顺序结构

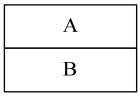
N-S 图的顺序结构表示方式如图 3-7 所示。

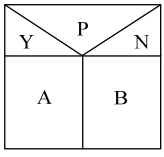
2）选择结构

N-S 图的选择结构表示方式如图 3-8 所示。

3）循环结构

N-S 图的循环结构表示方式如图 3-9 所示。





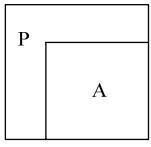


图 3-7　N-S 图的顺序结构表示方式　图 3-8　N-S 图的选择结构表示方式　图 3-9　N-S 图的循环结构表示方式

用 N-S 图表示算法与流程图相似，只是图形表示方式不同，图 3-10 是用 N-S 图来描述例 3.1 的算法，这里就不多做说明了。

除了以上讲述的几种表示方法外，还可以用介于自然语言和计算机语言之间的文字和符号即伪代码等来表示算法，我们最后编写得到的可以在计算机上执行的 C 程序其实也可以看成一种算法表示方法。

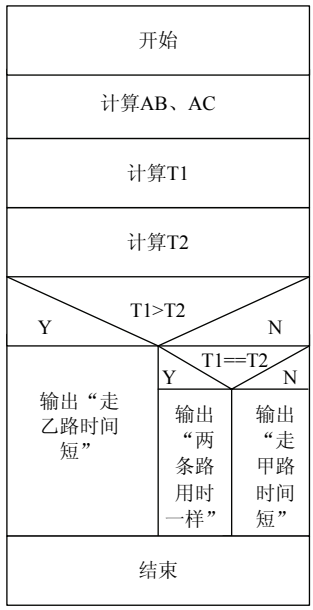


图 3-10　例 3.1 的 N-S 流程图

3.2　C 语句概述

本节将讲述 C 语句，使读者对 C 程序有一个初步的认识，为后面的编程打下基础。

任何 C 程序都可以看成由预编译指令、数据声明、执行语句组成。C 编译系统首先对预编译指令进行预处理，由预处理得到的结果与程序其他部分一起，组成可以用来编译的最后的源程序，然后由编译程序对该源程序正式进行编译，得到目标程序。声明部分用来定义所使用的变量、函数等，它不产生机器指令，而执行部分就是由 C 语句组成的，程序的功能由执行语句实现的。

C 语句有多种，可分为表达式语句、控制语句、函数调用语句、复合语句、空语句等。下面讲解各种语句的定义格式、功能以及在使用中应注意的问题。

3.2.1　表达式语句

表达式语句是指在任何一种表达式末尾加上分号“;”所组成的语句，执行表达式语句就是计算表达式的值。在 C 语言程序中，表达式语句出现得最多。

其一般形式为：

```
表达式;
```

例如：

```
x=y+z;          /*赋值语句*/
y+z;            /*加法运算语句，但计算结果不能保留，无实际意义*/
i++;            /*自增 1 语句，i 值增 1*/
```

要注意到表达式语句与表达式的区别，虽然仅仅差个分号，但二者是截然不同的。例如：在循环语句的条件中，要求用表达式作为条件，如果写成表达式语句，则是错误的。初学者一定要分清楚表达式和表达式语句的使用地方。

3.2.2　控制语句

控制语句用于控制程序语句的执行次序，它由特定的语句定义符组成。C 语言有九种控制语句。可分成以下三类：

（1）分支语句：if 语句、switch 语句。

（2）循环语句：while 语句、do while 语句、for 语句。

（3）转向语句：break 语句、goto 语句、continue 语句、return 语句。

这些语句将在以后的章节中逐一学习。

3.2.3　函数调用语句

一个 C 程序通常是由多个函数组成的，每个函数会完成一个基本功能。C 语言还提供了一些能完成某些常用功能函数，用户也可以编写更多自己的函数。这些函数在 C 程序使用时，需要被调用，这是由函数调用语句来完成的。函数调用语句由函数名、实际参数及分号组成。

函数调用语句一般形式为：

```
函数名(实际参数表);
```

例如：

```
printf("C  Program");          //格式输出函数，输出字符串
```

执行函数调用语句时，会把调用函数的实际参数传递给函数定义中的形式参数，然后执行被调函数体中的语句，将计算结果返回给调用函数。关于函数内容，将在后面章节中为读者做详细的介绍。

3.2.4　复合语句

上面说过的类似 i++;这样的语句，是单条语句。如果将两条或两条以上的语句用花括号{}括起来组成一个语句就构成一个复合语句，例如：

```
{x=y+z;
printf("%d", x);
}
```

复合语句内的各条语句都必须以分号结尾，在结束花括号“}”外不能加分号。在程序中应把复合语句看成单条语句，而不是多条语句。

3.2.5　空语句

空语句是一种有分号而没有表达式的特殊语句，即它只由一个分号“;”组成，执行空语句时将什么也不做。

空语句在编程中也是有用的，例如：它可用来做循环体，也就是通常所说的空循环。下面是一个为了延迟一段时间的循环，其循环体为空语句。

```
for(i=0; i<1000:i++)
;
```

3.3　数据的输入/输出

一个功能相对独立的程序包括三部分：第一部分为变量提供数据（数据输入），第二部分为计

算处理部分，第三部分为结果输出。从前面的例子中，可以看到几乎每个程序都包含输入/输出操作，没有输入/输出过程的程序是没有实际意义的。

所谓输入就是将数据通过键盘、扫描仪等设备将信息传递给计算机的过程，计算机将处理完的数据输出到显示器、打印机的过程就是输出。

C 语言本身中没有输入/输出语句，而是由 C 标准库函数中的输入/输出函数完成的，这些函数包含在 stdio.h 头文件，在程序中使用标准输入/输出库函数时，必须包含预编译命令#include，将 stdio.h 头文件包含进来，完整的预编译命令如下：

```
#include < stdio.h >
```

或

```
#include "stdio.h"
```

其中，stdio 是 standard input & output 的意思。两种编译命令的区别是，#include <stdio.h>的查找文件路径为编译系统的子目录，而#include "stdio.h"首先查找用户当前目录，当找不到时，再查找编译系统子目录。

3.3.1 格式输出函数 printf()

printf()函数称为格式输出函数，字母 f 为英文 format 即格式的缩写。printf()函数的功能是，将指定的表达式的值按指定的格式输出到屏幕上。在前面的学习中已多次使用了这个函数。在这里，我们将对其做详细的介绍。

printf()函数调用的一般形式为：

```
printf("格式控制字符串"，输出表列)
```

功能：按格式字符串中的格式依次输出列表中的各数据。

从 printf()函数的格式可以看出，其参数由格式控制字符串、输出表列两部分组成。格式控制字符串用于指定数据的输出样式，输出表列中指出需要输出的各项变量或表达式，各变量和表达式间用逗号分隔。

例如：

```
printf("How are you\n");                /*输出：How are you 并换行*/
printf("r=%d,s=%f\n",2,3.14*2*2);       /*输出：r=2,s=12.560000*/
```

在格式控制字符串中，可以是非格式字符串或者格式字符串。非格式字符串在输出时原样输出，在显示中起提示作用。

例如：`printf("hello! ");`

该语句的运行结果是在屏幕上显示 hello！字符串，格式控制字符串“hello!”就属于非格式字符串。

1．格式字符串

格式字符串则以百分号%开头，其后跟格式字符，在两者之间可以有选择地添加各种格式说明符，以说明输出数据的形式、长度、小数位数等。不同的格式控制串对应不同类型数据的输出。例如：“%d”表示按十进制整型格式输出。

格式字符的一般形式：% [附加格式说明符] 格式符

例如：%d，%10.2f 等。其中，%d 格式符表示以十进制整型格式输出，而%f 表示以实型格式输出，附加格式说明符“10.2”表示输出宽度为 10，输出 2 位小数。常用的格式说明符见表 3-1，附加格式说明符见表 3-2。

表 3-1　常用的格式说明符

格式符	功　能	例　如	结　果
d,i	输出带符号十进制整数	int a=567; printf ("%d",a);	567
o	输出无符号八进制整数（无前缀 o）	int a=65; printf("%o",a);	101
x,X	输出无符号十六进制整数(无前缀 ox)	int a=255; printf("%x",a);	ff
u	输出无符号整数	int a=567; printf("%u",a);	567
c	输出单个字符	char a=65; printf("%c",a);	A
s	输出一串字符	printf("%s", "ABC");	ABC
f	输出实数（6 位小数）	float a=567.789; printf("%f",a);	567.789001
e,E	以指数形式输出实数（尾数含 1 位整数，6 位小数，指数至多 3 位）	float a=567.789; printf("%e",a);	5.677890e+002
g,G	选用 f 与 e 格式中输出宽度较小的格式，且不输出无意义的 0	float a=567.789; printf("%g",a);	567.789

说明：

①格式符除 X、E、G 外，必须使用小写字母，否则无效。

例如：printf("%D, %%d", 10, 12)；输出%D, %%d 而不是 10, 12。

对允许大写格式的 X、E、G，输出的结果数据中含有字母的，字母用大写表示。

如 int a=255; printf("%X",a);则结果为 FF。

②格式符与输出项个数应相同，按先后顺序一一对应。

③输出转换：格式字符与输出项类型不一致，自动按指定格式输出。

表 3-2　附加格式说明符

附加格式说明符	功　能
m（m 为正整数）	数据输出宽度为 m，数据长度<m，左补空格，否则按实际输出
.n（n 为正整数）	对实数，n 是输出的小数位数，对字符串，n 表示输出前 n 个字符
-	数据左对齐输出，缺省时右对齐输出
+	指定在有符号数的正数前显示正号（+）
0	输出数值时指定左面不使用的空位置自动填 0
#	在八进制和十六进制数前显示前导 0，0x
l	在 d,o,x,u 前，指定输出 long 型数据，在 e,f,g 前，指定输出 double 型数据

参照下面的例题理解附加格式说明的意义。

【例 3.4】输出数据时使用 m.n。

```
#include<stdio.h>
 void main(void)
 {  int a=1234;
    float f=123.456;
    char ch='a';
    printf("12345678901234567890123456789012345678901234567890\n");
    printf("%8d,%2d\n",a,a);
    printf("%f,%8f,%8.1f,%.2f,%.2e,%g\n",f,f,f,f,f,f);
    printf("%3c\n",ch);
}
```

运行结果如图 3-11 所示。

```
12345678901234567890123456789012345678901234567890
    1234,1234
123.456001,123.456001,   123.5,123.46,1.23e+002,123.456
  a
Press any key to continue
```

图 3-11　例 3.4 运行结果

分析：理解屏幕输出位置可参照程序中先输出的一行“1234567890123...”，以下程序相同。

【例 3.5】输出字符串时使用 m.n。

```
#include<stdio.h>
 void main(void)
 {
  static char a[]="Hello,world!";
  printf("12345678901234567890123456789012345678901234567890\n");
  printf("%s\n%15s\n%10.5s\n%2.5s\n%.3s\n",a,a,a,a,a);
 }
```

运行结果如图 3-12 所示。

```
12345678901234567890123456789012345678901234567890
Hello,world!
   Hello,world!
     Hello
Hello
Hel
Press any key to continue
```

图 3-12　例 3.5 运行结果

【例 3.6】输出数据时使用-。

```
#include<stdio.h>
 void main(void)
 {
    int a=1234;
    float f=123.456;
    static char c[]="Hello,world!";
    printf("12345678901234567890123456789012345678901234567890\n");
    printf("%8d,%-8d#\n",a,a);
    printf("%10.2f,%-10.1f#\n",f,f);
    printf("%10.5s,%-10.3s#\n",c,c);
}
```

运行结果如图 3-13 所示。

```
12345678901234567890123456789012345678901234567890
    1234,1234    #
    123.46,123.5     #
     Hello,Hel       #
Press any key to continue
```

图 3-13　例 3.6 运行结果

【例 3.7】输出数据时使用 0、+。

```
#include<stdio.h>
```

```
void main(void)
{
   int a=1234;
   float f=123.456;
   printf("12345678901234567890123456789012345678901234567890\n");
   printf("%08d\n",a);
   printf("%010.2f\n",f);
   printf("%0+8d\n",a);
   printf("%0+10.2f\n",f);
}
```

运行结果如图 3-14 所示。

```
12345678901234567890123456789012345678901234567890
00001234
0000123.46
+0001234
+000123.46
```

图 3-14　例 3.7 运行结果

【例 3.8】输出数据时使用#。

```
#include<stdio.h>
void main(void)
{
   int a=123;
   printf("12345678901234567890123456789012345678901234567890\n");
   printf("%o,%#o,%X,%#x\n",a,a,a,a);
}
```

运行结果如图 3-15 所示。

```
12345678901234567890123456789012345678901234567890
173,0173,7B,0x7b
```

图 3-15　例 3.8 运行结果

总结：格式说明串的一般形式和意义，如图 3-16 所示。

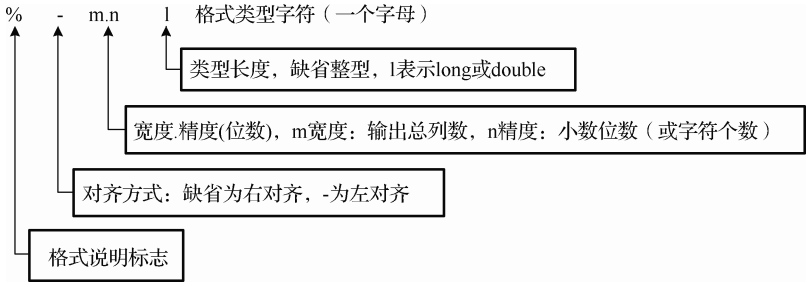


图 3-16　格式说明串的一般形式和意义

【例 3.9】printf()函数格式字符串输出。

```
main()
{
   int a=12345,b=15;
```

```
    char c='p';
    float d=12.345;
    printf("a=%d\n",a);
    printf("b=%5d,%o\n",b,b);
    printf("c=%5c,%c\n",c,c);
    printf("d=%f\n",d);
    printf("d=%8.2f\n",d);
    printf("d=%.2f\n",d);
}
```

```
a=12345
b=   15,17
c=    p,p
d=12.345000
d=   12.35
d=12.35
Press any key to continue
```

图 3-17　例 3.9 运行结果

运行结果如图 3-17 所示。

分析：第 7 行“%5d”限定了变量 b 以固定长度 5 输出，b 值为 15，只有 2 位，故补 3 个空格，“%o”要求变量 b 以八进制输出。第 10 行“%8.2f”指定输出宽度为 8，由于指定精度为 2，小数位数超过 2 位部分被截去，前面补 3 个空格。第 11 行“%.2f”没有指定输出宽度，仅限定了精度。

在输出格式控制字符串中，我们经常会使用类似“\n”的字符，这样的字符被称为转义字符，可以控制输出数据的位置。下面介绍 C 语言中的一些转义字符。

2．转义字符

如果要输出不可打印字符，则需要在格式控制字符串中使用转义序列。所谓转义序列是指在格式控制字符串中用该字符的 ASCII 码值来表示输出，其格式为“\ddd”，即反斜线后面跟该字符 3 位八进制 ASCII 码值，也可用十六进制数字表示字符的 ASCII 码值，其格式为“\xhh”。

因为用 ASCII 码来表示输出不可打印字符使用起来会不方便，所以 C 语言将一些常用的控制字符的转义字符用“\字母”形式来表示，常用的控制代码的转义字符如下。

\n——换行。

\t——水平制表。

\b——退格。

\'——单引号。

\r——回车。

\\——反斜线。

3.3.2　格式输入函数 scanf()

scanf()函数称为格式输入函数，即按指定的格式从键盘上把数据输入到指定的变量中。

scanf()函数的一般形式为：

```
scanf("格式控制字符串", 地址表列);
```

功能：按格式字符串中规定的格式，从键盘输入读取输入的数据，并依次赋给各输入项。

其中，格式控制字符串的作用与 printf()函数相同，指出输入数据的输入格式。地址表列是指各变量的地址，由地址运算符“&”后跟变量名组成。例如：&a，表示变量 a 的地址。所谓变量的地址是指数据在计算机系统中的存放的位置，读者不必关心具体的地址是多少。对于初学者，使用 scanf()函数时，比较常见的错误是忘记在变量前面加上地址变量符&。

scanf()函数格式控制字符串的一般形式为：

%[*][输入数据宽度][长度]类型

scanf 的格式字符：d,i,o,x,u,c,s,f,e,g 等，功能如表 3-3 所示。

表 3-3　scanf 的格式字符

格 式 字 符	功　　能
d,i	用于输入有符号十进制数
u	用于输入无符号十进制数
o	用于输入无符号八进制数
X,x	用于输入无符号十六进制数
c	用于输入单个字符
s	用于输入字符串
f	用于输入实数，以小数形式或指数形式
E,e,g,G	用于输入实数，与 f 作用相同

scanf 附加格式说明符（修饰符）如表 3-4 所示。

表 3-4　scanf 附加格式说明符

修　饰　符	功　　能
h	用于 d、o、x 前，指定输入为 short 型整数
l	用于 d、o、x、u 前，指定输入为 long 型整数；用于 e、f 前，指定输入为 double 型实数（必需）
m	指定输入数据宽度，遇空格或不可转换字符则结束
*	抑制符，指定输入项读入后不赋给变量

输入方法如下。

（1）格式字符串中包含的普通字符必须按原样输入。

```
scanf ("%d,%d",&a,&b);
```

若输入序列为 2,3↙，则 a=2, b=3。

若输入序列为 12□13↙（□表示空格），则 a 和 b 均不能得到预期的数据。

（2）截取输入数据。

```
scanf("%d,%4d",&a,&b );
```

若输入序列为 1.23,12345↙，则 a=1.23, b=1234。虽然输入的是 12345，但%4d 宽度为 4 位，截取前 4 位，即 1234。

（3）输入数据时，遇以下情况认为该数据输入结束。

①遇空格、跳格（Tab 键）或回车。

②遇宽度结束。

③遇非法输入。

例如：
```
int a,b,d; char c;
scanf("%d%d%c%3d",&a,&b,&c,&d);
```

若输入序列为 10□11A12345↙，则 a=10, b=11, c='A', d=123。

10 后的空格表示数据 10 输入结束；11 后遇字符'A'，对数值变量 b 而言是非法的，故数字 11 到此结束；而'A'对应 c；最后一个数据对应的宽度为 3，故截取 12345 前三位 123。注意，输入 b 数据 11 后不能用空格结束，这是因为下一个数据为一字符，而空格也是字符，将被变量 c 接受，则 c 的值不是'A'而是空格。

（4）输入字符时的情况。

使用“%c”格式符时，空格和转义字符作为有效字符输入，例如：

```
scanf("%c%c%c",&c1,&c2,&c3);
```

若输入 a□b□c↙，则 a⇒c1, □ ⇒c2, b ⇒c3，c2 为空格，可输入 abc↙。

（5）附加格式说明符 m 可以指定数据宽度，但不允许用附加格式说明符.n（例如用.n 规定输入的小数位数）。

例如：scanf("%10.3f,%10f,%f",&a,&b,&c);其中，%10.3f 是错误的。

（6）输入 double 数据时必须用%lf 或%le。在 printf 函数中输出 double 型数据可以用%f 或%e，但输入 double 型数据时必须用%lf 或%le。

（7）附加格式说明符"*"允许对应的输入数据不赋给相应变量。

例如：scanf("%4d%2d%2d",&yy,&mm,&dd);

若输入 19991015↙（↙表示回车键），则 1999⇒yy, 10 ⇒mm, 15 ⇒dd。

```
scanf("%3c%2c",&c1,&c2);
```

若输入 abcde↙，则'a'⇒c1, 'd' ⇒c2。

```
scanf("%3d%*4d%f",&k,&f);
```

若输入 12345678765.43↙，则 123⇒k, 8765.43⇒f。

再如，

```
double a;int b;float c;
  scanf("%lf,%2d,%*d,%5f",&a,&b,&c);
```

在键盘上输入：7.4,24,567,1. 42345↙

输入后，a 的值为 7.4，b 的值为 24，c 的值为 1.423。%*d 对应的数据是 567，因此 567 实际未赋给 c 变量，把 1.42345 按%5f 格式截取 1.423 赋给 c。

【例 3.10】scanf()函数格式字符串输入。

```
main()
{
    int a,b,c;
    printf("input a,b,c\n");
    scanf("a=%d,b=%d,c=%d",&a,&b,&c);
    scanf("%d,%d,%d",&a,&b,&c);
    scanf("%d%d%d",&a,&b,&c);
    printf("a=%d,b=%d,c=%d",a,b,c);
}
```

```
input a,b,c
a=2,b=3,c=4
2,3,4
2 3 4
a=2,b=3,c=4
Press any key to continue
```

图 3-18 例 3.10 运行结果

运行结果如图 3-18 所示。

分析：如果要将数据 2 赋给变量 a，3、4 赋给变量 b、c，对于第 4、5、6 行 3 个不同的输入函数，在输入数据时，其输入格式是不同的。第 4 行的输入格式为 a=2, b=3, c=4，第 5 行为 2, 3, 4，第 6 行为 2 3 4。所以，在向变量中输入数据时，要严格按照格式控制字符串中的格式，否则变量是不能获得正确值的。

要注意，在输入多个数值数据时，如果在格式控制串中没有非格式字符作为输入数据之间的间隔，则可用空格、Tab 或回车作为间隔。比如上例中的第 4 行语句，在输入时，在每个数据间加了空格作为分隔，如果直接输入 234，将只有变量 a 获得 234，而变量 b、c 是没有被赋值的。

另外，在输入字符数据时，要注意 C 语言将所有输入的字符均视为有效字符。例如：

```
scanf("%c%c%c",&a,&b,&c);
```

这里我们要将字符 a、b、c 分别赋给变量 a、b、c，当输入格式为 a b c 时，注意这里在每个字符间加了空格，则赋值结果为变量 a、b、c，得到的字符依次为 a、b，要注意变量 b 被赋值的字符是空格，只有当连续输入 abc 三个字符时，才能得到正确赋值结果。

3.3.3　字符输出函数 putchar()

putchar()函数称为字符输出函数，其功能是将所指定的一个字符输出到屏幕上，即将该字符显示在屏幕上。该函数的格式形式为：

```
putchar(c);
```

其中，c 是该函数的参数。参数 c 可以是一个字符常量，也可以是字符型变量，还可以是表达式。正常情况下，该函数返回输出字符的 ASCII 代码值。出错时，返回 EOF。使用时要添加头文件 stdio.h，其中的参数 ch 是要进行输出的字符，可以是字符型变量或整型变量，也可以是常量。

例如：

```
putchar('A');           /*输出大写字母 A*/
putchar(x);             /*输出字符变量 x 的值*/
putchar('\101');        /*也是输出字符 A*/
putchar('\n');          /*换行*/
```

【例 3.11】使用 putchar()函数实现字符数据输出。

在程序中使用 putchar()函数，输出字符串“Hello”并且在字符串输出完毕之后进行换行。

```
#include<stdio.h>
int main()
{
     char cChar1,cChar,2,cChar3,cChar4, cChar5;
     cChar1='H';
     cChar2='e';
     cChar3='l';
     cChar4='l';
     cChar5='o';
     putchar(cChar1);
     putchar(cChar2);
     putchar(cChar3);
     putchar(cChar4);
     putchar(cChar5);
     putchar('\n');
     return 0;
}
```

```
Hello
```

图 3-19　例 3.11 运行结果

运行结果如图 3-19 所示。

分析：要使用 putchar()函数，首先要包含头文件 stdio.h。声明字符型变量，用来保存要输出的字符。

为字符变量赋值，因为 putchar()函数只能输出一个字符。如果要输出字符串，就需要多次调用 putchar()函数。

当字符串输出完毕之后，使用 putchar()函数输出转义字符\n 进行换行操作。

3.3.4 字符串输出函数 puts()

puts()函数称为字符串输出函数，其功能是将所指定的字符串显示在屏幕上。该函数的格式形式为：

```
puts(s);
```

其中，s 是该函数的参数，参数 s 指出要输出显示的字符串。它可以是一个字符串常量，也可以是一个字符型数组，或是一个指向字符串的指针。该函数正常时返回零。

使用该函数时，先要在其程序中添加 stdio.h 头文件。其中，形式参数 str 是字符指针类型，可以用来接收要输出的字符串。例如，使用 puts()函数输出一个字符串：

```
puts("I LOVE CHINA!");
```

这行语句是输出一个字符串，之后会自动进行换行操作。这与 printf()函数有所不同，在前面的实例中使用 printf()函数进行换行时，要在其中添加转义字符'\n'。puts()函数会在字符串中判断"\0"结束符，遇到结束符时，后面的字符不再输出并且自动换行。例如:

```
puts("I LOVE\0 CHINA!");
```

在上面的语句中加"\0"字符后，puts()函数输出的字符串就变成：I LOVE。

分析：编译器会在字符串常量的末尾添加结束符"\0"，这也就说明了 puts()函数会在输出字符串常量时最后进行换行操作的原因。

【例 3.12】使用字符串输出函数显示信息提示。

在本例中，使用 puts()函数对字符串常量和字符串变量都进行操作，在这些操作中观察使用 puts()函数的方式。

```
#include <stdio.h>
int main()
{
    Char *Char="ILOVECHINA";          /*定义字符串指针变量*/
    puts("ILOVECHINA!");              /*输出字符串常量*/
    puts("I\0LOVE\0CHINA!");          /*输出字符串常量，其中加入结束符"\0"*/
    puts(Char);                       /*输出字符串变量的值*/
    Char="ILOVE\0CHINA!";             /*改变字符串变量的值*/
    puts(Char);                       /*输出字符串变量的值*/
    return 0;                         /*程序结束*/
}
```

运行结果如图 3-20 所示。

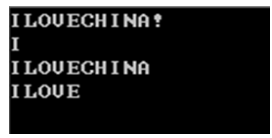


图 3-20 例 3.12 运行结果

分析：在程序代码中可以看到字符串常量赋值给字符串指针变量，有关字符串指针的内容将会在后面的章节中进行介绍。此时，可以将其看成整型变量，为其赋值后，就可以使用该变量。

第一次使用 puts()函数输出字符串常量，由于在该字符串中没有结束符"\0"，所以输出的字符会一直到最后编译器为其字符串添加的结束符"\0"为止。

在第二次使用 puts()函数输出的字符串常量中，为其添加两个"\0"。输出的显示结果表明检测字符时，如果遇到第一个结束符便不再输出字符并进行换行操作。

第三次使用 puts()函数输出的是字符串指针变量，函数根据变量的值进行输出。因为在变量的值中并没有结束符，所以会一直将字符输出到最后编译器为其添加的结束字符，然后进行换行操作。

改变变量的值，再使用 puts()函数输出变量时，可以看到由于变量的值中有结束符"\0"，因此显示结果到第 1 个结束符后停止，最后进行换行操作。

3.3.5　字符输入函数 getchar()

getchar()函数是用键盘输入的函数，该函数的功能是从键盘上获取一个字符，该函数的格式形式为：

```
getchar();
```

该函数没有参数，返回值是所接收的字符的 ASCII 码值。

【例 3.13】使用 getchar()函数实现字符数据输入。

在本实例中，使用 getchar()函数获取在键盘上输入的字符，再利用 putchar()函数进行输出。注意：本实例是将 getchar()作为 putchar()函数表达式的一部分，来进行输入和输出字符的。

```
#include <stdio.h>
int main()
{
    char cChar1;              /*声明变量*/
    cChar1=getchar();         /*在输入设备得到字符*/
    putchar(cChar1);          /*输出字符*/
    putchar('\n');            /*输出转义字符换行*/
    getchar();                /*得到回车字符*/
    putchar(getchar());       /*得到输入字符，直接输出*/
    putchar('\n');            /*换行*/
    return 0;                 /*程序结束*/
}
```

图 3-21　例 3.13 运行结果

运行结果如图 3-21 所示。

分析：要使用 getchar()函数，首先要包括头文件 stdio.h。声明变量 cChar1 通过 getchar 得到输入的字符，赋值给 cChar1 字符型变量。然后，使用 putchar()函数将变量进行输出。使用 getehar()函数得到输入过程中的回车符。在 putchar()函数的参数位置调用 getchar()函数得到字符，将得到的字符输出。

【例 3.14】使用 getchar()函数取消获取回车符。

```
#include <stdio.h>
int main()
{
    char  cChar1;             /*声明变量*/
    cChar1=getchar();         /*在输入设备得到字符*/
    putchar(cChar1);          /*输出字符*/
    putchar('\n');            /*输出转义字符换行*/
                              /*将此处 getchar 函数删掉*/
    putchar(getchar());       /*得到输入字符，直接输出*/
    putchar('\n');            /*换行*/
    return 0;                 /*程序结束*/
}
```

运行结果如图 3-22 所示。

图 3-22　例 3.14 运行结果

分析：在程序中将 getchar()函数获取回车符的语句去掉，比较两个程序的运行情况。从程序的显示结果可以发现，程序没有获取第二次的字符输入，而是进行了两次回车操作。

3.3.6 字符串输入函数 gets()

gets()函数是字符串输入函数，其功能是从键盘上获取所输入的字符串。该函数的格式形式为：

```
gets(s);
```

其中，s 是该函数参数，其意义与 puts()函数相同，该函数的正常返回值是一个字符型指针即读取到的字符串的首地址，出错时返回 NULL。

【例 3.15】gets()函数输入字符串。

```
#include <stdio.h>
main()
{printf("input a string\n");
char s[10];
gets(s);
puts(s);
}
```

```
input a string
hello
hello
Press any key to continue
```

图 3-23 例 3.15 运行结果

运行结果如图 3-23 所示。

分析：例 3.16 是利用 gets()函数接收从键盘输入字符串到字符数组 s 中，再利用 puts()函数将字符数组中字符输出。关于字符数组的内容将在后面章节中介绍。

【例 3.16】使用字符串输入函数 gets()获取输入信息。

```
#include <stdio.h>
main()
{
    char  cString[30];          /*定义一个字符数组变量*/
    gets(cString);              /*获取字符串*/
    puts(cString);              /*输出字符串*/
    return 0;                   /*程序结束*/
}
```

```
I LOVE CHINA!
I LOVE CHINA!
```

图 3-24 例 3.16 运行结果

运行结果如图 3-24 所示。

分析：因为要接收输入的字符串，所以要定义一个可以接收字符串的变量。在程序代码中，定义 cString 为字符数组变量的标识符。关于字符数组的内容将在后面的章节中进行介绍，此处知道此变量可以接收字符串即可。

用 gets()函数，其中函数的参数为定义的 cString 变量。调用该函数时，程序会等待用户输入字符，当用户字符输入完毕按 Enter 键确定时，gets()函数获取字符结束。使用 puts 字符串输出函数将获取后的字符串进行输出。

3.4 顺序结构程序设计举例

程序执行的顺序按照语句书写顺序从前向后顺次执行，这种结构称为顺序结构。仅含有顺序结构的程序特点是算法简单，只能解决最简单的问题。

【例 3.17】输入三角形的三边长，求三角形面积。

已知三角形的三边长 a，b，c，则该三角形的面积公式为：

$$area = \sqrt{s(s-a)(s-b)(s-c)}$$

其中 s = (a+b+c)/2。

```
#include<math.h>
main()
{
    float a,b,c,s,area;
    scanf("%f,%f,%f",&a,&b,&c);
    s=1.0/2*(a+b+c);
    area=sqrt(s*(s-a)*(s-b)*(s-c));
    printf("a=%5.2f,b=%5.2f,c=%5.2f,s=%5.2f,area=%5.2f\n",a,b,c,s,area);
}
```

运行结果如图 3-25 所示。

```
2,3,4
a= 2.00,b= 3.00,c= 4.00,s= 4.50,area= 2.90
Press any key to continue
```

图 3-25　例 3.17 运行结果

分析：本例可以实现的功能是通过输入三角形三边来计算三角形面积，其中用到了 sqrt()函数。该函数可以完成开平方运算，相应地使用这类函数时，要通过预编译指令 include 将包含其的头文件 math.h 包含进来。

这是一个仅含有顺序结构的程序，如果从键盘输入的三条边为 2，3，4，不能构成三角形，但程序依然会计算三角形的面积，会出现一个错误的结果。能否在输入三角形三条边后进行判断：是否构成三角形？如果能构成三角形，则计算该三角形面积；如不能，则输出不能构成三角形的信息？这是可以的，可以采用后面章节中介绍的选择结构。

【例 3.18】编写一个程序，提示用户从键盘输入一个星期的薪水和工作时数，它们均为浮点数，计算并输出每小时的平均薪水，输出格式为：您每小时的平均薪水收入为*元*毛*分。

```
#include<stdio.h>
main()
{
    float s,h,aveg;
    int yuan,jiao,fen;
    printf("请输入您一个星期的薪水和工作时数,用空格分隔\n");
    scanf("%f%f",&s,&h);
    aveg=s/h;
    yuan=(int)aveg;
    jiao=(int)((aveg-yuan)*10);
    fen=(int)(((aveg-yuan)*10-jiao)*10);
    printf("您每小时的平均薪水收入为%d 元%d 毛%d 分。\n",yuan,jiao,fen);
}
```

运行结果如图 3-26 所示。

```
请输入您一个星期的薪水和工作时数,用空格分隔
1200 40
您每小时的平均薪水收入为30元0毛0分。
```

图 3-26　例 3.18 运行结果

【例 3.19】计算圆的面积。

在本例中，定义单精度浮点型变量，为其赋值为圆周率的值。得到用户输入的数据并进行计算，最后将计算的结果输出。

```
#include<stdio.h>
main()
{
    float  Pie=3.14;
    float  Area;
    float  Radius;
    puts("Enter the radius:");
    canf("%f",&Radius);
    Area=Radius*Radius*Pie;
    printf("The Area is: %.2f\n",Area);
    return 0;
}
```

```
Enter the radius:
5
The Area is: 78.50
```

图 3-27　例 3.19 运行结果

运行结果如图 3-27 所示。

分析：定义单精度浮点型 Pie 表示圆周率，变量 Area 表示圆的面积，变量 Radius 表示圆的半径。

根据 puts()函数输出的程序提示信息，使用 scanf()函数输入半径的数据，将输入的数据保存在变量 Radius 中。

圆的面积=圆的半径的平方×圆周率。运用该公式，将变量放入其中计算圆的面积，最后使用 printf()函数将结果输出。在 printf()函数中可以看到%.2f 格式关键字，其中的 2 表示取小数点后两位。

【例 3.20】将大写字符转化成小写字符。

本例要将一个输入的大写字符转化成小写字符，需要对其中的 ASCII 码的关系有所了解。将大写字符转化成小写字符的方法就是将大写字符的 ASCII 码转化成小写字符的 ASCII 码。

```
#include<stdio.h>
main()
{
    char  cBig;
    char  cSmall;
    puts("Please enter capital character:");
    cBig=getchar();
    puts("Minuscule character is:");
    cSmall=cBig+32;
    printf("%c\n",cSmall);
    return 0;
}
```

```
Please enter capital character:
T
Minuscule character is:
t
```

图 3-28　例 3.20 运行结果

运行结果如图 3-28 所示。

分析：为了将大写字符转化为小写字符，要为其定义变量并进行保存。cBig 表示要存储字符的字符变量，而 cSmall 表示要转化成的小写字符。

通过信息提示，用户输入字符。因为只要得到一个输入的字符即可，所以在此处使用 getchar()函数就可以满足程序的要求。

大写字符与小写字符的 ASCII 码值相差 32。例如字符 A 的 ASCII 值为 65，a 的 ASCII 值为 97，因此如果要将一个大写字符转化成小写字符，那么将大写字符的 ASCII 值加上 32 即可。

字符变量 cSmall 得到转化的小写字符后，利用 printf 格式输出函数将字符进行输出，其中使用的格式字符为%c。

本 章 小 结

本章介绍了许多基础知识，包括算法的概念及表示方式，语句和常用的数据输入及输出等。每一个程序都有一定功能，为解决一个问题而采取的方法和步骤就是算法，在 C 语言中常用程序流程图等方式来表示算法。在使用 scanf()函数时，常忘记地址运算符&等，读者可能不会一次掌握全部内容，可以在以后的学习中复习本章内容。

习　题　3

一、程序题

1．当运行以下程序时，在键盘上从第一列开始输入 9876543210↙（此处↙代表回车），运行程序输出结果。

```
main( )
{
    int a; float b,c;
    scanf(" %2d%3f%4f",&a,&b,&c);
    printf(" \na=%d,b=%f,c=%f\n",a,b,c);
}
```

2．若定义 double a,b,c;要求为 a、b、c 分别输入 10、20、30。输入序列为：（□表示空格）。

```
□10.0□□20.0□□30.0↙
```

则正确的输入语句是_____。

3．执行下列程序，按指定方式输入（□表示空格），判断能否得到指定的输出结果。若不能，请修改程序，使之能得到指定的输出结果。

输入：2□3□4↙

输出：a=2,b=3,c=4
　　x=6,y=24

程序：

```
main ()
{
    int a, b, c, x, y;
    scanf ("%d, %d, %d", a, b, c);
    x=a*b; y=x*c;
```

```
    printf ("%d %d %d", a, b, c);
    printf ("x=%f\n",x, "y=%f\n",y);
}
```

4．用下面的 scanf()函数输入数据，使 a=3, b=2, c='c'，d='d'。问：如何输入怎样才能给各变量正确赋值，并给出输出结果？

```
#include<stdio.h>
main()
{
    int a,b;
    char c,d;
    scanf("a=%d,b=%d\n",&a,&b);
    scanf("%c%c\n ",&a,&b);
    printf("%d,%d\n",a,b);
    printf("c=%d, d=%d,%c,%c,%c\n",c,d,c-32,d-32);
}
```

二、编程题

1．编程定义 int 类型的变量，初值为 97，依次按字符、十进制、八进制、十六进制格式输出该变量的值。

2．编写程序，输入时间 10:27 并把它转换成分钟后输出（从零点整开始计算）。

3．设圆半径为 7，编写程序，求圆的周长和面积。

4．编写程序，要求用*号输出字母 C 的图案。

5．编写程序，计算小王可以得到多少钱。

小王去银行存款，有本金 6000 元，他的计划是先存 4 年定期，余下的年份存活期。问：按照这种计划，5 年后，小王可以得到多少钱？已知 4 年定期存款利息为 9.68%，活期存款利息为 1.72%，要求结果保留到小数点后两位。

6．编写程序，提示用户从键盘输入一个星期的薪水和工作时数，它们均为浮点数，计算并输出每小时的平均薪水，输出格式为：您每小时的平均薪水收入为 20 元 8 毛 3 分。

第 4 章　选择结构程序设计

4.1　为什么需要选择结构程序设计

之前，我们学习了顺序结构的程序设计，我们知道顺序结构的程序的执行特点是，按照语句的书写顺序逐句执行，即执行完上一条语句后就自动执行下一条语句。但是在程序设计中，我们会碰到这样的情况：对一个条件进行判断，根据条件判断的不同结果，采取相应的方法处理。显然，顺序结构的程序设计无法解决上述的问题。然而，选择结构程序却能很好地解决上述问题。

选择结构是程序设计的三大基本结构之一，大多数程序中都会包含选择结构。它的作用是根据所指定的条件是否满足，决定从给定的两组操作中选择其一。实现选择结构的关键是判断所给的条件的结果。

在现实生活中，需要进行判断和选择的情况是很多的。

- 在解锁手机时，如果绘制正确的屏幕图案密码，则会进入手机主菜单界面（需要判断屏幕图案密码是否正确）。
- 在 ATM 机取款时需要验证交易密码（需要验证密码是否正确）。
- 如果考试不及格，则要补考（需要判断分数是否大于或等于 60）。
- 如果遇到红灯，则要停车等待（需要判断是否是红灯）。
- 周末不下雨，我们去郊游（需要判断天气是否晴好）。
- 身高 1.2m 以下的儿童或 65 岁以上的老人乘坐公交车免费（需要判断年龄或身高）。

条件判断不仅仅存在于日常生活中，由于程序需要处理的问题往往比较复杂，因此，在大多数程序中都会包含条件判断。学习程序设计，要善于分析条件，善于设计条件判断。

条件判断的结果是一个逻辑值:“是”或者“否”。例如:“今天你过生日吗”？答案只能是“是”或“否”，而不会是其他答案。再如：验证解锁手机时绘制的屏幕密码，结果只有两种，即“正确”或者“不正确”。在计算机语言中，用“真”和“假”来表示“是”或者“否”这两种状态。例如：判断一人能否免费乘坐公交车，我们会判断他的身高和年龄，如果是一位 70 岁的老人，我们就会得到“身高 1.2m 以下的儿童或 65 岁以上的老人”这个条件满足的结论，即条件为“真”。如果是一位身高 185cm 的男士乘车，我们会说条件不满足，即条件为“假”。

在 C 语言中，能实现选择结构程序设计的语句有 if 条件语句和 switch 多分支语句。在使用这些条件结构的语句时，会用到一些新的运算符和表达式来帮助我们清晰准确地描述条件，下面介绍两种运算符和表达式。

4.2　关系运算符和关系表达式

4.2.1　关系运算符

关系运算符：用来对两个数值进行比较的比较运算符。

C 语言提供以下 6 种关系运算符。

①<（小于运算符）。

②<= （小于或等于运算符）。

③>（大于运算符）。

④>= （大于或等于运算符）。

⑤= =（等于运算符）。

⑥!= （不等于运算符）。

要注意：关系运算符都是双目运算符，用于两个运算对象的比较。不能将“<=”写成“=<”，也不能将“>=”写成“=>”。

4.2.2 关系表达式

关系表达式：用关系运算符将两个数值或数值表达式连接起来的式子。

关系表达式的运算对象可以是常量，也可以是变量，还可以是表达式。例如：

```
a+b>c-d
x>3/2
'a'+1<c
-i-5*j==k+1
```

都是合法的关系表达式。由于表达式也可以又是关系表达式。因此，也允许出现嵌套的情况。例如：

```
a>(b>c)
a!=(c==d)
```

关系表达式的值是一个逻辑值，即“真”或“假”。例如，a>3 是一个关系表达式，大于号是一个关系运算符，如果 a 的值为 5，则满足给定的“a>3”条件，因此关系表达式的值为“真”（即“条件满足”）；如果 a 的值为 2，不满足“a>3”条件，则称关系表达式的值为“假”。因为 C 语言中没有逻辑类型数据，所以 C 语言规定用数值 0 代表“逻辑假”，用数值 1 代表“逻辑真”。因此，关系表达式的值只能是 1 或 0，其数据类型为整型。

设有定义 int a=3,b=2,c=1;则：

```
a>b      的值为 1
c==a     的值为 0
```

4.2.3 关系运算符的优先次序和结合性

1．关系运算符的优先次序

前 4 种关系运算符（<，<=，>，>=）的优先级别相同，后 2 种（==，!=）也相同。但是，前 4 种关系运算符的优先级要高于后 2 种。例如，“>”优先于“==”，而“>”与“<”的优先级相同。例如：

```
a>b==c 等价于(a>b)==c
```

关系运算符的优先级低于算术运算符。例如：

```
b*c>=10 等价于(b*c)>=10
```

关系运算符的优先级高于赋值运算符。例如：

```
a=b==c 等价于 a=(b==c)
```

关于算术运算符、关系运算符和赋值运算符的优先级次序为：

算术运算符　　高
关系运算符　　↓
赋值运算符　　低

例如：假设 n1=5，n2=8，n3=10，则：

（1）关系表达式 n1= =n2 的值为 0。

（2）关系表达式 n1<=n3 的值为 1。

（3）关系表达式 n1+n2>n3 的值为 1。

（4）关系表达式 n1= = n2>n3 的值等为 0。

2．结合方向

关系运算符的结合方向都为左结合，即运算符优先级相同时自左向右运算。例如：

```
a>=b>=c  等价于  (a>=b)>=c
```

根据运算符的优先级和结合方向，表达式 3==4+1>2!=5 的求解过程为：

①算术运算 4+1 的结果为 5（得表达式 3==5>2!=5）。

②关系运算 5>2 的结果为 1（得表达式 3==1!=5）。

③关系运算 3==1 的结果为 0（得表达式 0!=5）。

④关系运算 0!=5 的结果为 1，即整个表达式的结果为 1。

注意：

（1）“=”和“= =”是不同的两类运算符，前者为赋值运算符，是对变量进行赋值运算；后者为关系运算符，强调判断运算符“==”前后的操作数是否相等，会得到一个“是”或“否”的答案。例如：

①x=5;

②x==5;

上述语句①表示把变量 x 赋值为 5，而语句②表示的含义是判断变量 x 的值是否为 5，表达式的值取决于变量 x 的值。若 x 的值为 5，则整个表达式的值是 1；若 x 的值不等于 5，则表达式的结果为 0。

（2）在使用关系运算符时，应避免对实型数据做相等或不相等的判断。例如：1.0/3.0*3.0 = =1.0 的结果为 0，可改为 fabs(1.0/3.0*3.0-1.0)<1e-6。

（3）从本质上说，关系运算的结果（即关系表达式的值）不是数值，而是逻辑值。但是，由于 C 语言没有提供逻辑型数据（C99 增加了逻辑型数据，用关键字 bool 定义逻辑型变量），为了处理关系运算和逻辑运算的结果，C 语言指定用 1 代表真，0 代表假，并在编译系统中按此实现。因此，使用 C 语言的人就要遵守这样的规定。由于用了 1 和 0 代表真和假，而 1 和 0 又是数值，所以在 C 程序中还允许把关系运算的结果（即 1 和 0）视为和其他数值型数据一样，可以参加数值运算，或把它赋给数值型变量。例如，若 a、b、c 的值 a=1，b=2，c=3，请分析下面的赋值表达式：

d=a>b　d 的值为 0

f=a>b>c　f 的值为 0（因为“>”运算符的结合方向是自左至右，先执行“a>b”，得值 0，再执行关系运算“0>c”，得值 0，赋给 f）。

关系运算符的结合性都是自左向右的。使用关系运算符的时候常常会判断两个表达式的关系。但是，由于运算符存在优先级的问题，因此如果不小心处理则会出现错误。例如，要进行这样的判断操作：先对一个变量进行赋值，然后判断这个赋值的变量是否不等于一个常数，代码如下：

```
if(Number=NewNum!=10)
{
...
}
```

因为“!=”运算符比“=”的优先级要高，所以 NewNum!=10 的判断操作会在赋值之前实现，变量 Number 得到的就是关系表达式的真值或者假值，这样并不会按照之前的意愿执行。对此可以使用括号来表示要优先计算的表达式，例如：

```
if((Number=NewNum)!=10)
{
...
}
```

这种写法比较清楚，不会产生混淆，没有人会对代码的含义产生误解。由于这种写法格式比较精确简洁，因此被多数的程序员所接受。

C 语言所表现的这种灵活性使初学者理解起来较困难，容易出错。初学者应注重于程序的清晰性和易读性，不要编写易错、复杂、别人不容易理解的程序。

4.3 逻辑运算符和逻辑表达式

有时，要求判断的条件不是一个简单的条件，而是由几个给定简单条件组成的复合条件。

例如：“如果星期六不下雨，我去公园玩。”这就是由两个简单条件组成的复合条件，需要判定两个条件：（1）是否星期六；（2）是否下雨。只有这两个条件都满足，才去公园玩。

又如：“招聘年龄在 25～35 岁之间的教师”，这就需要检查两个条件：（1）年龄 age>=25（2）年龄 age<=35。这个组合条件是不能够用一个关系表达式来表示的，要用两个表达式的组合来表示，即 age>=25 AND age<=35。用一个逻辑运算符 AND 连接 age>=25 和 age<=35。两个关系表达式组成一个复合条件。“AND”的含义是“与”，即“二者同时满足”。age>=25 AND age<=35 表示 age>=25 和 age<=35 同时满足。这个复合的关系表达式“age>=25 AND age<=35”就是一个逻辑表达式。其他逻辑表达式可以有：

```
x>0 AND y>0          （同时满足 x>0 和 y>0）
age<12 OR age>65     （年龄 age 小于 12 的儿童或大于 65 的老人）
```

上面第 1 个逻辑表达式的含义是：只有 x>0 和 y>0 都为“真”时，逻辑表达式 x>0 AND y>0 才为“真”。上面第 2 个逻辑表达式的含义是：age<12 或 age>65 至少有一个为“真”时，逻辑表达式 age<12 OR age>65 为“真”。OR 是“或”的意思，即“有一即可”，在两个条件中有一个满足即可。AND 和 OR 是逻辑运算符。

用逻辑运算符将关系表达式或其他逻辑量连接起来的式子就是逻辑表达式。

4.3.1 逻辑运算符

C 语言提供了三个逻辑运算符。

（1）&&：逻辑“与”运算符。

逻辑“与”（&&），类似于两个表达式做乘积。乘积的结果值只有两种：如果结果值非 0，则为逻辑“真”；如果结果值为 0，则为逻辑假。例如，3&&5 的结果值为“真”。逻辑“与”运算符要求参与运算的两个表达式同为“真”，结果才为“真”，这种关系类似于图 4-1 中的两个开关，只有当两个开关同时闭合时灯泡才会亮。

（2）||：逻辑“或”运算符。

逻辑“或”(||)，类似于两个表达式做累加和。如果结果值非 0，则值为“真”；如果结果值为 0，则值为假。例如 3||0=1。逻辑“或”运算符参与运算的两个表达式有一个为“真”时，结果就为“真”，这种关系类似于图 4-2 的两个开关，只要有一个开关闭合灯泡就会亮。

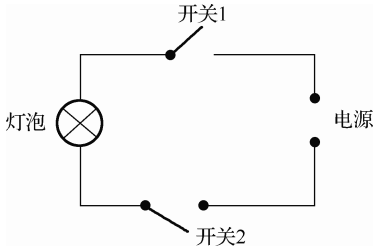


图 4-1　逻辑“与”运算举例电路图

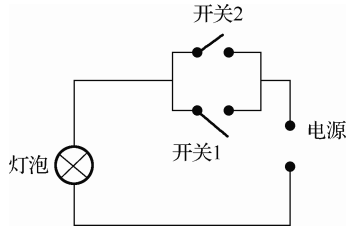


图 4-2　逻辑“或”运算举例电路图

（3）!：逻辑“非”运算符。

逻辑“非”(!)，它是取反操作。真值取反，结果值为“假”；假值取反，结果值为“真”。例如!0=1。

以上逻辑运算符中，“&&”和“||”是双目运算符，它们要求有两个运算量，如 a&&b，(x>y)||(y>z)。“!”是单目运算符，即只要求有一个运算量，如!x。特别需要注意的是，C 语言规定，所有非 0 值均认为是“真”值，即为 1，只有 0 认为是“假”值，即为 0。

4.3.2　逻辑表达式

用逻辑运算符将运算对象连接起来的式子称为逻辑表达式。与关系运算符类似，逻辑表达式的值也有两种，即“真”和“假”。运算对象一般为关系表达式或逻辑量（常量或变量）。

如果 a、b 为运算对象，则逻辑运算符的运算规则如下。

（1）若 a、b 都为“真”，则 a&&b 为“真”，否则为“假”。

（2）若 a、b 都为“假”，则 a||b 为“假”，否则为“真”。

（3）若 a 为“真”，则!a 为“假”；若 a 为“假”，则!a 为“真”。

针对 a 与 b 可能取不同值，逻辑运算结果如表 4-1 所示。

表 4-1　逻辑运算真值表

a	b	!a	!b	a&&b	a||b
真	真	假	假	真	真
真	假	假	真	假	真
假	真	真	假	假	真
假	假	真	真	假	假

如前所述，逻辑表达式的值应该是一个逻辑量“真”或“假”。C 语言编译系统在表示逻辑运算结果时，以数值 1 代表“真”，以 0 代表“假”，但在判断一个量是否为“真”时，以 0 代表“假”，以非 0 代表“真”。即将一个非零的数值认为“真”。例如：

（1）若 a=4，则！a 的值为 0。因为 a 的值为非 0，被认为是“真”，对它进行“非运算”，得“假”。“假”以 0 代表。

（2）若 a= –4，b=5，则 a&&b 的值为 1。因为 a 和 b 均为非 0，被认为是“真”，因此，a&&b 的值也为“真”，值为 1。

（3）a 和 b 值分别为 4 和 5，a ||b 的值为 1。

（4）a 和 b 值分别为 4 和 5，！ a || b 的值为 1。

（5）4&&0.02 的值为 1。

通过这几个例子可以看出，由系统给出的逻辑运算结果不是 0 就是 1，不可能是其他数值。而在逻辑表达式中作为参加逻辑运算的运算对象可以是 0（“假”）或任何非 0 的数值（按“真”对待）。如果在一个表达式中不同位置上出现数值，应区分哪些作为数值运算或关系运算的对象，哪些作为逻辑运算的对象。例如：

```
5>3 && 8<4 -!0
```

表达式自左至右扫描求解。首先处理“5>3”（因为关系运算符优先于逻辑运算符&&）。在关系运算符“>”两侧的 5 和 3 作为数值参加关系运算，“5>3”的值为 1（代表真）。再进行“1&&8<4-!0”的运算，8 的左侧为“&&”，右侧为“<”运算符，根据优先规则，应先进行“<”的运算，即先进行“8<4-!0”的运算。现在 4 的左侧为“<”，右侧为“–”运算符，而“–”优先于“<”，因此应先进行“4–!0”的运算，由于“!”的级别最高，因此先进行“!0”的运算，得到结果 1。然后进行“4-1”的运算，得到结果 3，再进行“8<3”的运算，得 0，最后进行“1&&0”的运算，得 0。

实际上，逻辑运算符两侧的运算对象不但可以是 0 和 1，或者是 0 和非 0 的整数，也可以是字符型、浮点型、枚举型或指针型的纯量型数据。系统最终以 0 和非 0 来判定它们是“真”或“假”。例如：'a'&&'d'的值为 1（因为'a'和'd'的 ASCII 值都不为 0，按“真”处理），所以 1&&1 的值为 1。

在逻辑运算中，并非所有运算都必须进行。只有在必须求出下一个表达式才能求出逻辑表达式的值时，才进行相关运算。我们需掌握如下两种特殊的逻辑运算。

（1）a&&b&&c。它的计算过程如图 4-3 所示。

（2）a||b||c。它的计算过程如图 4-4 所示。

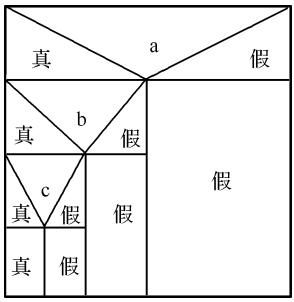


图 4-3 a&&b&&c 运算 N-S 图

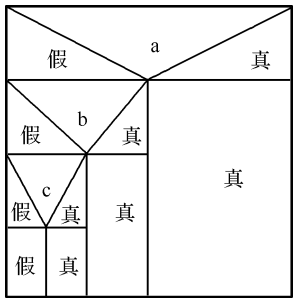


图 4-4 a||b||c 运算 N-S 图

根据上面两个流程图，对于&&运算来说，只有 a 为真时，才进行以下的两个运算；而对于||运算来说，只有 a 为假时，才进行以下运算。

因此，如果有下面的逻辑表达式：

```
(m=a>b)&&(n=c-d)
```

当 a=l, b=2, c=3, d=4，m 和 n 的原值为 1 时，由于“a>b”的值为 0，因此 m=0，此时已能判定整个表达式不可能为真，不必再进行“n=c>d”的运算，因此 n 的值不是 0 而仍保持原值 1，请读者注意这一点。

4.3.3　逻辑运算符的优先次序和结合性

1．优先级

三个逻辑运算符的优先级，从高到低的顺序为：

```
！→ && → ||
```

在一个逻辑表达式中，如果包含多个逻辑运算符，如 a&&b||!c，则应该按照下面的优先顺序进行运算：逻辑“非”，逻辑“与”，逻辑“或”。

前面介绍了几种常用运算符的优先关系，加上逻辑运算符后，优先关系变为：

!（逻辑“非”）　　高
算术运算符
关系运算符
&&和||（逻辑“与”和逻辑“或”）
赋值运算符　　低

例如：3>2+1&&4<2+1　等价于　(3>(2+1))&&(4<(2+1))

2．结合方向

“&&”和“||”的结合方向为左结合，“!”的结合方向为右结合。例如：a>b&&c>d&&e>f 等价于((a>b)&&(c>d))&&(e>f)，!!!(a>b)等价于!(!(!(a>b)))。

熟练掌握 C 语言的关系运算符和逻辑运算符后，可以巧妙地用一个逻辑表达式来表示一个复杂的条件。

例如，判别用 year 表示的某一年是否是闰年。闰年的条件应符合下面二者之一：

（1）能被 4 整除，但不能被 100 整除，如 2008。

（2）能被 4 整除，又能被 400 整除，如 2000（注意，能被 100 整除但不能被 400 整除的年份不是闰年，如 2100）。

以上条件可以用一个逻辑表达式来表示：

```
year%400==0||(year%4==0&&year%100!=0)
```

当 year 为某一整数值时，如果上述表达式值为真（即为 1），则 year 为闰年；否则 year 为非闰年。可以用 N-S 图表示其逻辑结构，如图 4-5 所示。

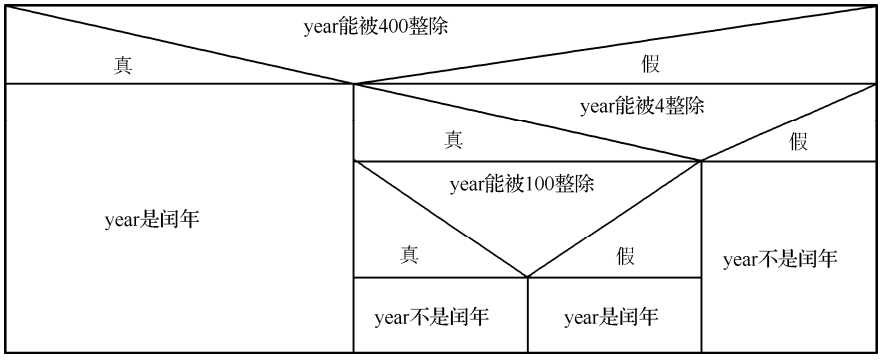


图 4-5　判断 year 是否为闰年的 N-S 图

4.4 用 if 语句实现选择结构

在日常生活中，为了使交通畅通有序，一般会在路口设立交通信号灯。在信号灯显示为绿色时，车辆可以行驶通过；当信号灯转为红色时，车辆就要停止行驶。可见，信号灯给出了信号，人们通过不同的信号进行判断，然后根据判断的结果进行相应的操作。

在 C 语言程序中，也可以完成这样的判断操作，利用的就是 if 语句。if 语句的功能就像路口的信号灯一样，根据判断不同的条件，决定是否进行操作。

据说第一台数字计算机是用来进行决策操作的，使得之后的计算机都继承了这项功能。程序员将决策表示成对条件的检验，即判断一个表达式值的真假。除了没有任何返回值的函数和返回无法判断真假的结构函数外，几乎所有表达式的返回值都可以判断真假。

下面具体介绍 if 语句的有关内容。

4.4.1 if 语句的基本形式

if 语句就是判断表达式的值，然后根据该值的情况控制程序流程。表达式的值不等于 0，也就是为“真”。否则，就是“假”值。if 语句有 3 种形式，分别为 if 语句形式、if…else 语句形式、else if 语句形式。每种情况的具体使用方式如下。

1. if 语句形式

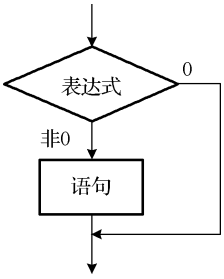


图 4-6 if 语句流程图

if 语句通过对表达式进行判断，根据判断的结果选择是否进行相应的操作。if 语句的一般形式为：

```
if(表达式) 语句
```

其语义是：如果表达式的值为真，则执行其后的语句，否则不执行该语句。其执行过程可表示为如图 4-6 所示。

例如下面的程序：

```
if(Num) printf("The true value");
```

在程序中判断变量 Num 的值，如果变量 Num 为真值，则执行后面的输出语句；如果变量的值为假，则不执行。

在 if 语句的括号中，不仅可以判断一个变量的值是否为真，也可以判断表达式，例如：

```
if(Signal==l)  printf("the Signal Light is%d",Signal);
```

这行代码的含义：判断变量 Signal==l 的表达式，如果 Signal==1 条件成立，那么判断的结果是真值，则执行后面的输出语句；如果条件不成立，那么结果为假值，则不执行后面的输出语句。

在这些代码中可以看到 if 后面的执行部分只是调用了一条语句，如果是两条语句，应该怎么办呢？这个时候，可以使用大括号使之成为语句块，例如：

```
if(Signal==l)
{   printf("the Signal Light is%d:\n",Signal);
    printf("Cars can run");
}
```

将执行的语句都放在大括号中，这样当 if 语句判断条件为真时，就可以全部执行。使用这种方式的好处是可以很规范、清楚地表达出 if 语句所包含语句的范围。因此，建议读者在使用 if 语句时都用大括号将执行语句包括在内。

【例 4.1】使用 if 语句模拟信号灯指挥车辆行驶。

在本例中，为了模拟十字路口上信号灯指挥车辆行驶，要使用 if 语句判断信号灯的状态。如果信号灯为绿色，则说明车辆可以行驶通过，通过输出语句进行信息提示说明车辆的行动状态。

```
#include<stdio.h>
main( )
{   int  Signal;                          /*定义变量表示信号灯的状态*/
   printf("the Red Light is 0, the Green Light is 1:\n"); /*输出提示信息*/
   scanf("%d",&Signal);                   /*输入信号灯状态*/
   if(Signal==1)                          /*使用 if 语句判断信号灯状态*/
   { printf("the Light is green,cars can run.\n"); }  /*判断结果为真时输出*/
}
```

运行结果如图 4-7 所示。

```
the Red Light is 0, the Green Light is 1:
1
the Light is green,cars can run.
```

图 4-7　例 4.1 运行结果

分析：

（1）为了模拟信号灯指挥交通，要根据信号灯的状态进行判断，这样就需要用一个变量表示信号灯的状态。在程序代码中，定义变量 Signal 表示信号灯的状态。

（2）输出提示信息，输入 Signal 变量，表示此时信号灯的状态，此时用键盘输入 1，表示信号灯状态是绿灯。

（3）使用 if 语句判断 Signal 变量的值。如果为真，则表示的信号灯为绿灯；如果为假，则表示的是红灯。在程序中，此时变量 Signal 的值为 1，表达式 Signal==1 这个条件成立，因此判断的结果为真值，从而执行 if 语句后面大括号中的语句。程序运行结果如图 4-7 所示。

if 语句不是只可以使用一次，而是可以连续使用进行判断的，继而根据不同条件的成立给出相应的操作。

例如在上面的实例程序中，可以看到虽然使用 if 语句判断信号灯状态 Signal 变量，但只是给出了判断是绿灯时执行的操作，并没有给出红灯时相应的操作。为了使得在红灯情况下也进行操作，需要再使用一次 if 语句判断为红灯时的情况。现在对上面的实例进行完善，实例如下。

【例 4.2】完善 if 语句模拟信号灯的使用。

原程序中仅对绿灯情况下做出相应的操作，为进一步完善信号灯为红灯时的操作，在程序中再添加一次 if 语句对信号灯为红灯时的判断，并且在条件成立时给出相应的操作。

```
#include<stdio.h>
main()
{  int  Signal;              /*定义变量表示信号灯的状态*/
   printf("the Red Light is 0,the Green Light  is 1:\n ");  /*输出提示信息*/
   scanf("%d",&Signal);      /*输入信号灯状态*/
   if( Signal==1)            /*使用 if 语句进行判断*/
   {printf("the Light is green,cars can run.\n");}
   if( Signal==0)            /*使用 if 语句进行判断*/
   { printf("the Light is red,cars can't run.\n");}
}
```

运行结果如图 4-8 所示。

```
the Red Light is 0,the Green Light  is 1:
 0
the Light is red,cars can't run.
```

图 4-8　例 4.2 运行结果

分析：

（1）在上一个实例程序的基础上进行修改，完善程序的功能。在程序中添加一个 if 判断语句，用来表示当信号灯为红灯时进行相应的操作。

（2）从程序的开始处来分析整个程序的运行过程。使用 scanf 函数输入数据，这次用户输入为 0，表示红灯。

（3）程序继续执行，第一个 if 语句判断 Signal 变量的值是否为 1，如果判断的结果为真，则说明信号灯为绿灯。因为 Signal 变量的值为 0，所以判断的结果为假，则不会执行后面语句中的内容。

（4）接下来是新添加的 if 语句，在其中判断 Signal 变量是否等于 0，如果判断成立为真，则表示信号灯此时为红灯。因为输入的值为 0，所以 Signal==0 条件成立，执行 if 后面的语句内容。程序运行结果如图 4-8 所示。

2．if else 语句形式

除了可以指定在条件为真时执行某些语句外，还可以在条件为假时执行另外一段语句。这在 C 语言中是利用 else 语句来完成的，其一般形式为：

```
if（表达式）
    语句块 1
else
    语句块 2
```

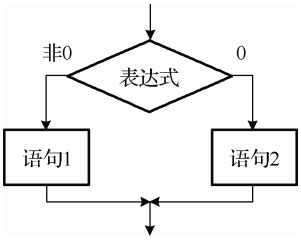


图 4-9　if...else 语句执行流程

其语句执行流程如图 4-9 所示。

在 if 后的括号中判断表达式的结果，如果判断的结果为真值，则执行紧跟 if 后的语句块中的内容；如果判断的为假值，则执行 else 语句后的语句块内容。也就是说，当 if 语句检验的条件为假时，就执行相应的 else 语句后面的语句或者语句块。例如下面的代码：

```
if(value)
{ printf("the value is true"); }
else
{printf("the value is false");}
```

在上面的代码中，如果 if 判断变量 value 的值为真，则执行 if 后面的语句块进行输出。如果 if 判断的结果为假值，则执行 else 下面的语句块。

注意：一个 else 语句必须跟在一个 if 语句的后面。

【例 4.3】输入两个整型数，将值较大者输出。

解题思路：专门设定一个变量 max，将较大者放到其中。这就需要对输入的两个整型数进行一次比较。如果 a 大于或等于 b，就把 a 的值赋给 max，否则就把 b 的值赋给 max，然后输出 max。其对应的 N-S 图如图 4-10 所示。

```
#include<stdio.h>
main()
{   int a,b,max;
    scanf("%d%d",&a,&b);
    if(a>=b)
       max=a;
    else
       max=b;
    printf("%5d\n",max);
}
```

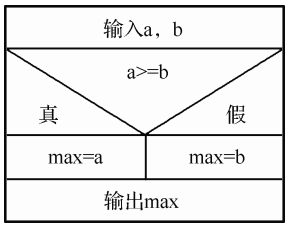


图 4-10　例 4.3 的 N-S 图

运行结果如图 4-11 所示。

```
3 8
   8
```

图 4-11　例 4.3 运行结果

分析：该程序的运行结果也可以用两个 if 语句来实现。读者可自行编写程序。本例中只使用了一个 if...else 语句，进行一次判断。显然，程序更简洁、清晰。程序运行结果如图 4-11 所示。

3．else if 形式

利用 if 和 else 关键字的组合可以实现 else if 语句，这是对一系列互斥的条件进行检验，其一般形式如下：

```
if(表达式 1)  语句 1
else  if(表达式 2)  语句 2
   else  if(表达式 3)  语句 3
         …
      else  if(表达式 n)  语句 n
         else 语句 m
```

根据流程图可知，首先对 if 语句中的表达式 1 进行判断，如果结果为真值，则执行后面跟着的语句 1，然后跳过 else if 语句和 else 语句；如果结果为假，那么判断 else if 中的表达式 2。如果表达式 2 为真值，那么执行语句 2 而不会执行后面 else if 的判断或者 else 语句。当所有的判断都不成立，也就是都为假值的时候，执行 else 后的语句块。由执行过程可知，n+1 个语句只有一个被执行，若 n 个表达式的值都为假，则执行语句 m，否则执行第一个表达式值为真（非 0）的后面的语句。

其语义是：依次判断表达式的值，当出现某个值为真时，则执行其对应的语句。然后，跳到整个 if 语句之外继续执行程序。如果所有的表达式均为假，则执行语句 m。然后继续执行后续程序。其执行过程可表示为如图 4-12 所示。

例如下面的程序：

```
if(Selection==l)
   {…}
    else if(Selection==2)
       {…}
          else if(Selection==3)
            {…}
```

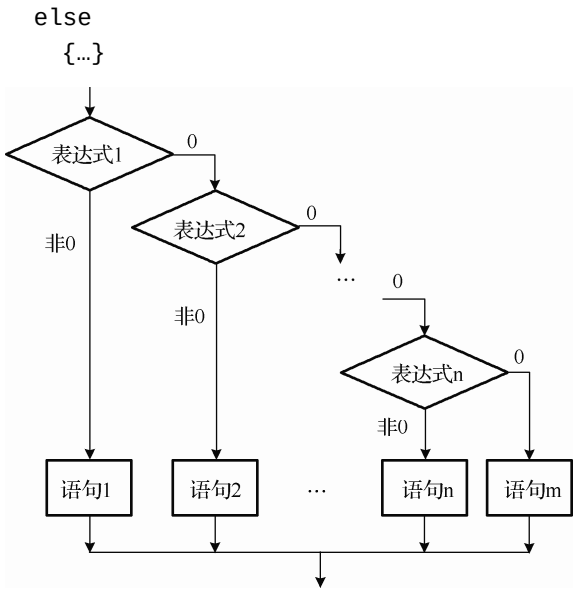


图 4-12　else if 语句的流程图

上述代码的含义是，使用 if 语句判断变量 Selection 的值是否为 1。如果为 1 则执行后面语句块中的内容，然后跳过后面的 else if 判断和 else 语句的执行；如果 Selection 的值不为 1，那么 else if 判断 Selection 的值是否为 2，如果值为 2，则条件为真执行后面紧跟着的语句块，执行完后跳过后面 else if 和 else 的操作；如果 Selection 的值也不为 2，那么接下来的 else if 语句判断 Selection 是否等于数值 3，如果等于则执行后面语句块中的内容，否则执行 else 的语句块中内容。也就是说，当前面所有的判断都不成立（为假值）时，执行 else 语句块中的内容。

在使用 if 语句时应注意以下问题。

（1）在两种形式的 if 语句中，if 关键字之后均为表达式。该表达式通常是逻辑表达式或关系表达式，但也可以是其他表达式，如赋值表达式等，甚至可以是一个变量。

（2）在 if 语句中，条件判断表达式必须用括号括起来，在语句之后必须加分号。

（3）在 if 语句中，所有的语句应为单个语句。如果要想在满足条件时执行一组（多个）语句，则必须把这一组语句用{ }括起来组成一个复合语句。

4.4.2　使用条件运算符改写 if 语句

有一种 if 语句，当被判别的表达式的值为“真”或“假”时，都执行一个赋值语句且向同一个变量赋值。例如：

```
if (a>b)
max=a;
else
max=b;
```

当 a>b 时将 a 的值赋给 max，当 a≤b 时将 b 的值赋给 max，可以看到无论 a>b 是否满足，都是给同一个变量赋值。C 语言提供条件运算符和条件表达式来处理这类问题。可以把上面的 if 语句改写为：

```
max= (a>b)?a:b;
```

条件运算符对一个表达式的真或假值结果进行检验，然后根据检验结果返回另外两个表达式中的一个。例如上面使用条件运算符的代码，首先判断表达式 a>b 是否成立，成立则说明结果为真，否则为假。当为真时，将 a 的值赋给 max 变量；如果为假，则将 b 的值赋给 max 变量。

【例 4.4】使用条件运算符计算欠款金额是否还清（本实例要求使用条件运算符进行判断选择）。

```
#include<stdio.h>
int main()
{
   float Dues=100;                          /*定义变量表示欠款金额为 100 元*/
   float Repayment;                         /*表示本次要还款金额*/
   printf("请输入还款金额:\n");              /*显示信息，提示输入还款金额*/
   scanf("%f",& Repayment);                 /*输入本次要还款金额*/
   (Repayment>Dues)?printf("本期账单欠款已还清"):printf("未还清本期账单");
   printf("应还款金额为%f\n",Dues-Repayment); /*显示还款后欠款金额*/
}
```

运行结果如图 4-13 所示。

图 4-13　例 4.4 运行结果

（1）在程序代码中，定义变量 Dues 表示欠款的金额，Repayment 表示还款的金额。

（2）通过运行程序时的提示信息，用户输入数据。假设用户输入还款的金额为 80，接下来使用条件运算符判断表达式还款金额>欠款金额（Repayment >Dues）是否成立，若成立则输出本期账单欠款已还清，若不成立则输出未还清本期账单。最后，还款后显示欠款金额。程序运行结果如图 4-13 所示。

4.5　选择结构的嵌套

当 if 语句中的执行语句又是 if 语句时，则构成了 if 语句嵌套的情形。

其一般形式可表示如下：

```
if(表达式)
   if 语句;
```

或者

```
   if(表达式)
      if 语句;
   else
      if 语句;
```

在嵌套内的 if 语句可能又是 if-else 型的，这将会出现多个 if 和多个 else 重叠的情况，这时要特别注意 if 和 else 的配对问题。

例如：

```
if(表达式 1)
   if(表达式 2)
```

```
        语句 1;
    else
        语句 2;
```

其中的 else 究竟是与哪一个 if 配对呢？

是应该理解为：

```
if(表达式 1)
    if(表达式 2)
        语句 1;
    else
        语句 2;
```

还是应该理解为：

```
if(表达式 1)
    if(表达式 2)
        语句 1;
else
        语句 2;
```

为了避免这种二义性，C 语言规定，else 总是与它前面最近的未配对的 if 配对，因此对上述例子应按第一种思路理解。其实，4.4 节中的 if 语句的第三种形式 else-if 形式就是一种 if 语句的嵌套，其形式如下：

```
if(表达式 1)  语句 1
else if(表达式 2)  语句 2
       else  if(表达式 3)  语句 3
             …
             else  if(表达式 n)  语句 n
                   else 语句 m
```

【例 4.5】已知分段函数如下。

$$y=\begin{cases}-1 & (x<0)\\ 0 & (x=0)\\ 1 & (x>0)\end{cases}$$

编写程序，输入 x，输出其对应的 y 值。

解题思路：

解此题的关键是写出 x 处于不同区间时 y 的不同取值。根据题意可以画出 N-S 流程图，如图 4-14 所示。

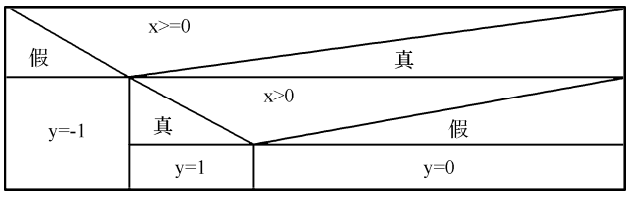


图 4-14　例 4.5 的 N-S 流程图

编写程序：采用嵌套的 if 语句处理。

程序 1：

```
main()
{   float x; int y;
    scanf("%f",&x);
    if(x>=0)
      if(x>0) y=1;
      else    y=0;
    else y=-1;
    printf("x=%f y=%d\n",x,y);
}
```

```
-2
x=-2.000000 y=-1
```

图 4-15　例 4.5 的运行结果

运行结果如图 4-15 所示。

此程序还可用其他方法实现，可将上面程序改写成如下形式。

程序 2：

```
# include <stdio. h>
int main()
{
    int x,y;
    scanf(“%d”,&x);
    if( x<0)
       y= -1;
       else
         if（x==0）y=0;
         else y=1;
    printf("x=%d,y=%d\n",x,y);
    return 0;
}
```

分析：程序 2 的执行结果和程序 1 的执行结果是相同的，但为了使逻辑关系清晰，避免出错，一般把内嵌的 if 语句放在外层的 else 子句中（如程序 2 那样），这样由于有外层的 else 相隔，内嵌的 else 不会被误认为与外层的 if 配对，而只能与内嵌的 if 配对，这样就不会搞混。程序运行结果如图 4-15 所示。

请读者弄清楚嵌套 if 中各个 if 的配对关系以及在程序中对嵌套 if 的书写格式。为了使程序清晰、易读，写程序时对选择结构和循环结构应采用锯齿形的缩进形式，如本书例题所示那样。

使用 if 语句时应注意以下几点：

（1）if 后面圆括号内的表达式可以为任意类型，但一般为关系表达式或逻辑表达式。

（2）if 和 else 后面的语句可以是任意语句。

（3）if(x)与 if(x!=0)等价。

（4）if(!x)与 if(x= =0)等价。

（5）多分支格式只是 if 语句嵌套格式的一种特例。

（6）else 总是与它上面最近的未配对的 if 配对。

4.6　用 switch 语句实现多分支选择结构

if 语句只有两个分支可供选择，而实际问题中常常需要用到多分支的选择。例如，人口统计分类（按年龄分为老、中、青、少、儿童）、工资统计分类、银行存款分类等。当然，这些都可以用嵌套的 if 语句来处理，但如果分支较多，则嵌套的 if 语句层数多，程序冗长且可读性降低。C 语言提供 switch 语句直接处理多分支选择。

switch 语句的一般形式如下：

```
switch(表达式)
{
case 常量 l：语句 1
case 常量 2：语句 2
...
case 常量 n：语句 n
default：    语句 n+l
}
```

其流程图如图 4-16 所示。

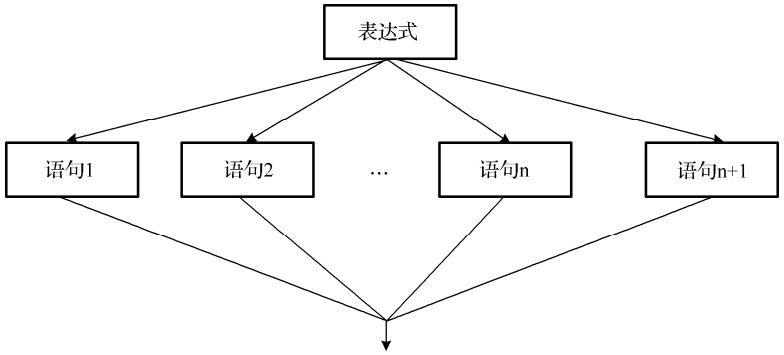


图 4-16　switch 语句流程图

通过上面的流程图分析 switch 语句的一般形式。switch 后面括号中的表达式就是要进行判断的条件。在 switch 的语句块中，使用 case 关键字表示检验条件符合的各种情况，其后的语句是相应的操作。其中还有一个 default 关键字，作用是，如果没有符合条件的情况，那么执行 default 后的默认情况语句。

说明：

（1）switch 后面的圆括号后不能加分号。

（2）switch 后面的圆括号内表达式的值必须为整型、字符型或枚举型。

（3）各 case 后面的常量表达式的值必须为整型、字符型或枚举型。

（4）各 case 后面的常量表达式的值必须互不相同。

（5）若某个 case 后面的常量表达式的值与 switch 后面的圆括号内表达式的值相等，就执行该 case 后面的语句，执行完后若未遇到 break 语句，则不再进行判断，接着执行下一个 case 后面的语句。若想执行完某一语句后退出，则必须在语句最后加上 break 语句。

（6）若每个 case 和 default 后面的语句都以 break 语句结束，则各个 case 和 default 位置可以互换。若每个 case 后包含 break，则顺序不影响最后的执行结果，反之则有可能影响执行。如下程序段

```
s=2;
switch (s)
{
   case 1: printf("1");break;
   case 2: printf("2");break;
   case 3: printf("3");break;
```

```
    default: printf("error");
}
```

的运行结果为 2，而如下程序段

```
switch (s)
{
   case 1: printf("1");
   case 2: printf("2");
   case 3: printf("3");
   default: printf("error");
}
```

的运行结果为 23error。

（7）case 后面的语句可以是任何语句，也可以为空，但 default 的后面不能为空语句。若为复合语句，则花括号可以省略。

（8）多个 case 可以共用一组语句。

```
switch (s)
{
   case 1:
   case 2:
   case 4:
   case 5: printf("%d",x);break;
}
```

本 switch 语句表示：s 的值为 1 或 2 或 4 或 5，都会执行 printf("%d",x);break;两条语句。也可以将其写成如下形式：

```
switch (s)
{
   case 1:  case 2:  case 4:  case 5:  printf("%d",x);break;
}
```

但不允许写成如下形式：

```
switch (s)
{
   case 1, 2, 4, 5:  printf("%d",x);break;
}
```

（9）switch-case 语句可以嵌套，即一个 switch-case 语句中又含有 switch-case 语句。但要注意 break 只能跳出最内层的 switch 语句。

【例 4.6】使用 switch 语句输出分数段，要求按照考试成绩的等级输出分数段，其中要使用 switch 语句判断分数的情况。

```
#include<stdio.h>
int main()
{
char cGrade;                                    /*定义变量表示分数的级别*/
printf("please enter your grade\n");            /*提示信息*/
scanf("%c",&cGrade);                            /*输入分数的级别*/
```

```
    printf("Grade is about:");                          /*提示信息*/
    switch(cGrade)                                      /*switch 语句判断*/
    {
    case 'A':                                           /*分数级别为 A 的情况*/
        printf("90～100\n");                            /*输出分数段*/
        break;                                          /*跳出*/
    case 'B':                                           /*分数级别为 B 的情况*/
        printf("80～89\n");                             /*输出分数段*/
        break;                                          /*跳出*/
    case 'C':                                           /*分数级别为 C 的情况*/
        printf("70～79\n");                             /*输出分数段*/
        break;                                          /*跳出*/
    case 'D':                                           /*分数级别为 D 的情况*/
        printf("60～69\n");                             /*输出分数段*/
        break;                                          /*跳出*/
    case 'F':                                           /*分数级别为 F 的情况*/
        printf("<60\n");                                /*输出分数段*/
        break;                                          /*跳出*/
    default:                                            /*默认情况*/
        printf("You enter the char is wrong!\n");       /*提示错误*/
        break;                                          /*跳出*/
    }
    return 0;
    }
```

```
please enter your grade
A
Grade is about:90~100
```

图 4-17　例 4.6 运行结果

运行结果如图 4-17 所示。

分析：

（1）在程序的代码中，定义变量 cGrade 用来保存用户输入的成绩并判定级别。

（2）使用 switch 语句判断字符变量 cGrade，其中使用 case 关键字检验可能出现的级别情况。并且在每一个 case 语句的最后都会有 break 进行跳出。如果没有符合的情况，则会执行 default 默认语句。在 case 语句表示的条件后有一个冒号“:”，在编写程序时不要忘记。

（3）在程序中，假设用户输入字符为'B'，在 case 检验中有为'B'的情况，那么执行该 case 的语句块，将分数段进行输出。运行程序，运行结果如图 4-17 所示。

【例 4.7】switch-case 语句的嵌套举例。

```
#include<stdio.h>
main()
{   int x,y,a=0,b=0;
    scanf("%d%d",&x,&y);
    switch(x)
    { case 1:
        switch(y)
         {
          case 0:a++;break;
          case 1:b++;break;
         }
      case 2:a++;b++;break;
```

```
        case 3:a++;b++;
      }
    printf("a=%d,b=%d\n",a,b);
}
```

```
1 0
a=2,b=1
```

图 4-18　例 4.7 运行结果

运行结果如图 4-18 所示。

分析：本程序中，switch(x)里面又嵌套了一个 switch 语句。程序需要录入 x 和 y 的值，当录入 x 的值为 1，y 的值为 0 时，switch(x)语句执行 case 1 后的语句，即执行 switch(y)中的 case 0 后的语句，a 自加一次后，执行 break 语句跳出 switch(y)语句。要注意的是，switch(y)语句后没有 break 语句，所以程序会继续执行 switch(x)中的 case 2 后面的语句，a 和 b 再自加一次后跳出 switch 语句，最后输出 a 和 b 的值，本例一定要注意内嵌 break 语句的作用。程序运行结果如图 4-18 所示。

4.7　选择结构程序设计举例

【例 4.8】使用 else if 编写屏幕菜单程序。

解题思路：既然要对菜单进行选择，那么首先要显示菜单。利用格式输出函数将菜单中所需要的信息进行输出。然后显示一条信息提示用户进行输入，选择一个菜单项进行操作。我们可以设置一个变量 iSelection 来存储用户的选择。最后使用 else if 语句根据 iSelection 不同取值输出不同菜单项。else if 语句流程图如图 4-19 所示。

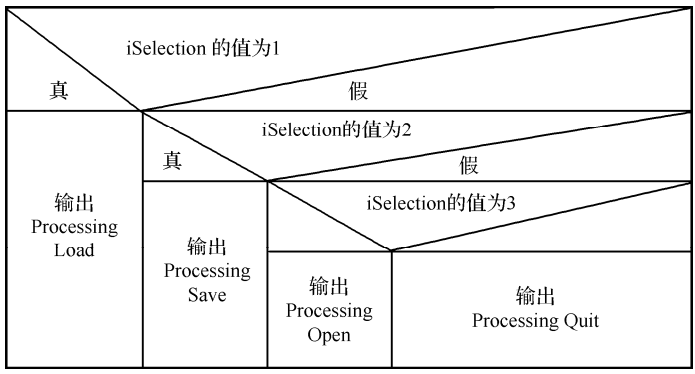


图 4-19　例 4.8 程序流程图

```
#include<stdio.h>
int main()
{
    int iSelection;                 /*定义变量，表示菜单的选项*/
    printf("---Menu---\n");         /*输出屏幕的菜单*/
    printf("1 = Load\n");
    printf("2 = Save\n");
    printf("3 = Open\n");
    printf("other = Quit\n");
    printf("enter selection\n");    /*提示信息*/
    scanf("%d",&iSelection);        /*用户输入选项*/
    if(iSelection==1)               /*选项为 1*/
        printf("Processing Load\n");
```

```
    else if(iSelection==2)          /*选项为 2*/
            printf("Processing Save\n");
        else if(iSelection==3)      /*选项为 3*/
                printf("Processing Open\n");
            else                    /*选项为其他数值*/
                printf("Processing Quit\n");
    return 0;
}
```

```
---Menu---
1 = Load
2 = Save
3 = Open
other = Quit
enter selection
2
Processing Save
```

图 4-20　例 4.8 运行结果

运行结果如图 4-20 所示。

分析：本程序首先使用多个 printf 函数输出菜单选项，然后提示用户输入选择的菜单编号，菜单编号存入变量 iSelection 中，菜单的选择是通过一个多分支的 if-else 语句实现的，根据 iSelection 的不同取值，进入相应的菜单项。程序运行结果如图 4-20 所示。

【例 4.9】从键盘上输入一个字符，请判断输入字符的种类，即判断它是数字字符、英文字符、空格、回车，还是其他字符。

解题思路：本题的关键在于如何判别字符的种类，字符在计算机中是以 ASCII 码的形式存储的，我们可以根据输入字符的 ASCII 码的范围确定字符种类。

算法可以用如图 4-21 所示 N-S 图描述。

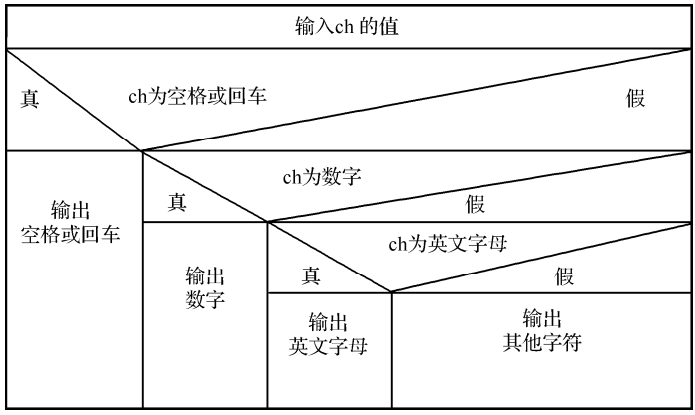


图 4-21　例 4.9 N-S 图

编写程序：

```
void main()
{
    char ch;
    printf("Input a character:");
    ch=getchar();
    if (ch==' '||ch=='\n')
            printf("This is a blank or enter.\n");
    else  if(ch>='0'&&ch<='9')
            printf("This ia a digit.\n");
         else  if(ch>='A'&&ch<='Z'||ch>='a'&&ch<='z')
                    printf("This ia a  letter.\n");
```

```
            else
                printf("This ia another character.\n");
    }
```

运行结果如图 4-22 所示。

```
Input a character:+
This ia another character.
```

图 4-22　例 4.9 的运行结果

分析：本程序声明字符型变量 ch 来存放输入的字符，使用多分支的 if-else-if 语句来完成对 ch 的种类的判断。若 ch 的 ASCII 码值跟空格或回车符相等，说明这是一个空格或回车；若 ch 的 ASCII 码值在'0'～'9'的 ASCII 码值之间，说明 ch 为数字；若 ch 的 ASCII 码值在'a'～'z'的 ASCII 码值之间或在'A'～'Z'的 ASCII 码值之间说明 ch 为英文字符；否则为其他字符。程序运行结果如图 4-22 所示。

【例 4.10】输入两个实数 a 和 b，再输入一个运算符（可以是+、-、*或/），根据运算符计算并输出 a、b 两个数的和、差、积、商。

编写程序：

```
#include<stdio.h>
void main()
{
   float a,b;
   char c;
   printf("please input a b:\n");
   printf("please choose +,-,* or / :\n");
   scanf("%f%f",&a,&b);
   c=getchar();
   switch(c)
   {
      case '+': printf("%f+%f=%f\n",a,b,a+b); break;
      case '-': printf("%f-%f=%f\n",a,b,a-b); break;
      case '*': printf("%f*%f=%f\n",a,b,a*b); break;
      case '/': if(b) printf("%f/%f=%f\n",a,b,a/b); break;
      default : printf("Can't compute!");
   }
}
```

```
please input a b:
please choose +,-,* or / :
5 8*
5.000000*8.000000=40.000000
```

图 4-23　例 4.10 的运行结果

运行结果如图 4-23 所示。

分析：本程序定义变量 a 和 b 来存放参与运算的数据，定义变量 c 来存放运算符号，输入 a 和 b 的值后，用户还要输入运算符号，switch 语句会根据变量 c 中的运算符号来选择相应的语句执行，最终输出结果。程序运行结果如图 4-23 所示。

【例 4.11】你买了一箱 n 个苹果，很不幸的是，买完时箱子里混进了一条虫子。虫子每 x 小时能吃掉一个苹果，假设虫子在吃完一个苹果之前不会吃另一个，那么经过 y 小时，你还有多少个完整的苹果？

```
#include <stdio.h>
int main()
{
   int n,x,y;   //一箱 n 个苹果，虫子每 x 小时吃掉一个苹果，经过 y 小时以后剩余的苹果
   int eaten;                              //eaten 为虫子吃掉的苹果
   scanf("%d%d%d",&n,&x,&y);
   if(y%x!=0)                              //虫子吃掉的苹果不为整数个
```

```
            eaten=y/x+1;                          //虫子吃剩的也不要了
        else                                      //虫子吃掉的苹果恰好为整数个
            eaten=y/x;                            //刚好吃完
        if(n>eaten)printf("%d\n",n-eaten);        //还有剩余的苹果
        else printf("%d\n",0);                    //苹果被全部吃光
        return 0;
    }
```

```
24 3 12
20
```

图 4-24　例 4.11 的运行结果

运行结果如图 4-24 所示。

分析：一箱 n 个苹果，虫子每 x 小时吃掉一个苹果。问：经过 y 小时以后剩余的完整苹果个数。设置变量 eaten 表示被吃的苹果数量，完整苹果个数为 n-eaten。但是，要注意考虑以下情况：若经过 y 小时后虫子吃完的苹果数量不是整数个，就是说虫子正在吃的苹果不能算成完整的苹果，即吃掉的苹果数量 eaten 为 y/x+1；若经过 y 小时后虫子吃完的苹果数量是整数个，吃掉的苹果数量 eaten 为 y/x；最后还要考虑经过 y 小时苹果是否被虫子全部吃光，若被虫子吃光则剩余 0 个。程序运行结果如图 4-24 所示。

本 章 小 结

通过本章的学习，要掌握关系运算符、逻辑运算符、条件运算符以及关系表达式、逻辑表达式和条件表达式的概念与用法。熟练掌握 if 语句和 switch 语句的使用。注意正确使用 if 语句的三种形式。在使用 switch 语句时，一定要注意：在没有 break 语句的情况下，case 语句的各个语句是逐条执行的，而不是执行一条语句就跳出 switch 语句。

选择结构是结构化程序设计的一个基本结构。它根据输入的数据或中间结果的情况，选择一组语句执行（在不同的情况下，选择不同的语句执行）。在编程时，必须将所有的情况都考虑进去，并写出在各种情况下所对应的语句组。学习程序设计，不要满足于能编写出程序，得到正确的结果，而应当力求编写出层次清晰、可读性好、书写美观的高质量的程序。

习　题　4

一、选择题

1．下列运算符中优先级最高的是（　　）。

A．>　　B．+　　C．&&　　D．!=

2．以下关于运算符优先级的描述中，正确的是（　　）。

A．!(逻辑非)>算术运算>关系运算>&&(逻辑与)>||(逻辑或)>赋值运算

B．&&(逻辑与)>算术运算>关系运算>赋值运算

C．关系运算>算术运算>&&(逻辑与)>||(逻辑或)>赋值运算

D．赋值运算>算术运算>关系运算>&&(逻辑与)>||(逻辑或)

3．逻辑运算符的运算对象的数据类型（　　）。

A．只能是 0 或 1　　B．只能是.T.或.F.

C．只能是整型或字符型　　D．任何类型的数据

4．能正确表示 x 的取值范围在[0, 100]和[-10, -5]内的表达式是（　　）。

A．(x<= –10)||(x>= –5)&&(x<=0)||(x>=100)

B．(x>= –10)&&(x<= –5)||(x>=0)&&(x<=100)

C．(x>= –10)&&(x<= –5)&&(x>=0)&&(x<=100)

D．(x<= –10)||(x>= –5)&&(x<=0)||(x>=100)

5．以下程序的运行结果是（　　）。

```
main()
{
   int  c,x,y;
   x=1;
   y=1;
   c=0;
   c=x++||y++;
   printf("\n%d%d%d\n",x,y,c);
}
```

A．110　　B．211　　C．011　　D．001

6．以下程序的运行结果是（　　）。

```
main()
{
   int  c,x,y;
   x=0;
   y=0;
   c=0;
   c=x++&&y++;
   printf("\n%d%d%d\n",x,y,c);
}
```

A．100　　B．211　　C．011　　D．001

7．判断字符型变量 ch 为大写字母的表达式是（　　）。

A．'A'<=ch<='Z'　　B．(ch>='A')&(ch<='Z')

C．(ch>='A')&&(ch<='Z')　　D．(ch>='A')AND(ch<='Z')

8．判断字符型变量 ch 为小写字母的表达式是（　　）。

A．'a'<=ch<='z'　　B．(ch>=a)&&(ch<=z)

C．(ch>='a')||(ch<='z')　　D．(ch>='a')&&(ch<='z')

9．以下 if 语句书写正确的是（　　）。

A．
```
if(x=0;)
    printf("%f",x);
  else  printf("%f",-x);
```

B．
```
if(x>0)
   {x=x+1; printf("%f",x);}
  else  printf("%f",-x);
```

C．
```
if(x>0)；
    {x=x+1; printf("%f",x);}
  else  printf("%f",-x);
```

D．
```
if(x>0)
    {x=x+1; printf("%f",x) }
  else  printf("%f",-x);
```

10．分析以下程序：

```
main()
{  int  x=5,a=0,b=0;
   if(x=a+b)    printf("** **\n");
   else         printf("## ##\n");
```

以上程序（　　）。

A．有语法错，不能通过编译　　B．通过编译，但不能连接

C．输出** **　　D．输出## ##

二、填空题

1．在 C 语言中，对于 if 语句，else 子句与 if 子句的配对约定是_____。

2．在 C 语言中提供的条件运算符“?:”的功能是_____。

3．条件表达式 a?b:c,其中 a,b,c 是三个运算分量。当运算分量 a 的值为真时，则_____，否则_____。

4．在 C 语言中的逻辑运算符的优先级是_____高于_____高于_____。

5．用 C 语言描述下列命题。

（1）a 小于 b 或小于 c_____

（2）a 和 b 都大于 c_____

（3）a 或 b 中有一个小于 c_ _____

（4）a 是奇数_____。

6．若 x=3,y=2,z=1，求下列表达式的值。

（1）x<y?y:x

（2）x<y?x++:y++

（3）z+=x<y?x++:y++

表达式的值分别是（1）_____；（2）_____；（3）_____。

7．表示条件：10<100 或 x<0 的 C 语言表达式是_____。

三、编程题

1．输入圆的半径 r 和一个整型数 k，当 k=1 时，计算圆的面积；但 k=2 时，计算圆的周长，当 k=3 时，既要求求出圆的周长也要求出圆的面积。编程实现以上功能。

2．有一函数，其函数关系如下，试编程求对应于每一自变量的函数值。

$$y=\begin{cases} x^2 & (x<0) \\ -0.5x+10 & (0\leqslant x<10) \\ x-\sqrt{x} & (x\geqslant 10) \end{cases}$$

第 5 章　循环结构程序设计

5.1　为什么使用循环结构

什么是循环结构？其实在日常生活中，我们经常碰到类似于循环结构思想的例子。例如，晨练时计划在操场上跑完 5 圈再回家。我们的头脑中想的就是，如果没到 5 圈，那么接着再跑一圈，直到够 5 圈了，结束晨练回家。再如，某旅行社组织的一次 30 人的云南丽江旅行，可接受 30 名团员报名，旅行社工作人员在每次接待报名人员时，都会查看此时报名者总人数，如果总数没到 30，那么就可以继续安排报名者填报个人信息和缴纳费用，这个动作会重复 30 次结束。这也是一种循环的思想。在日常的生活中会有许多简单而重复的工作，为完成这些必要的工作需要花费很多的时间，而编写程序的目的就是使工作变得简单，使用计算机来处理这些重复的工作是最好的方法。

循环结构是结构化程序设计中一种很重要的结构。其特点是，在给定条件成立时，反复执行某程序段，直到条件不成立为止。给定的条件称为循环条件，反复执行的程序段称为循环体。

C 语言提供了多种循环语句，可以组成各种不同形式的循环结构。

C 语言中实现循环的语句：

（1）while 语句。

（2）do…while 语句。

（3）for 语句。

下面分别介绍每一种语句的循环结构的使用。

5.2　用 while 语句实现循环结构程序设计

1．while 语句的一般形式

一般形式：

```
while（表达式）
    循环体语句
```

2．执行过程

while 语句首先检验一个条件，也就是括号中的表达式。当条件为真时，就执行紧跟其后的语句或者语句块。每执行一遍循环，程序都将回到 while 语句处，重新检验条件是否满足。如果一开始条件就不满足，则跳过循环体里的语句，直接执行后面的程序代码。如果第一次检验时条件满足，那么在第一次或其后的循环过程中，必须有使条件为假的操作，否则循环无法终止，执行过程如图 5-1 所示。

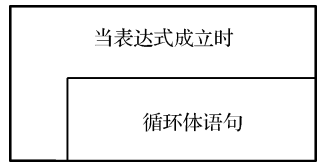


图 5-1　while 语句的执行过程

3．执行特点

首先判断表达式，然后执行语句。

【例 5.1】用 while 循环求 1+2+3+…+100。

解题思路：

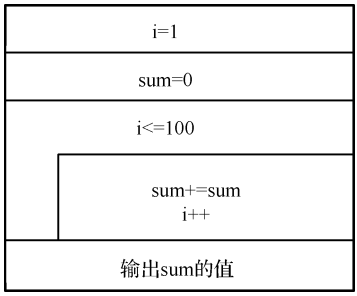


图 5-2 例 5.1 程序的 N-S 图

（1）这是一个累加的问题，需要先后将 100 个数相加。要重复进行 100 次加法运算。

（2）分析每次所加的数有无规律？发现每次累加的数是有规律的，后一个数是前一个数加 1。因此，不需要每次用 scanf() 函数从键盘临时输入数据，只要在加完上一个数 i 后，使 i 加 1 就可得到下一个数。

也可以用循环结构来实现。重复执行循环体 100 次，每次加一个数。

算法用 N-S 图表示，如图 5-2 所示。

```
#include<stdio.h>
int main()
{
    int Sum=0;                          /*定义变量，表示计算总和*/
    int i=1;                            /*表示每一个数字*/
    while(i<=100)                       /*使用 while 循环*/
    {
        Sum=Sum+i;                      /*进行累加*/
        i++;                            /*增加数字*/
    }
    printf("the result is：%d\n",Sum); /*将结果输出*/
    return 0;
}
```

```
the result is: 5050
```

图 5-3 例 5.1 程序运行结果

运行结果如图 5-3 所示。

程序分析：

（1）循环体如果包含一个以上的语句，应该用花括号括起来，作为复合语句出现。如果不加花括号，则 while 语句的范围只到 while 后面第 1 个分号处。例如，本例中 while 语句中如无花括号，则 while 语句范围只到“sum= sum+i;”。

（2）不要忽略给 i 和 sum 赋初值（这是未进行累加前的初始情况），否则它们的值是不可预测的，结果显然不正确。读者可上机试一下。

（3）在循环体中应有使循环趋向于结束的语句。例如，在本例中循环结束的条件是“i>100”，因此在循环体中应该有使 i 增值以最终导致 i>100 的语句，今用“i++;”语句来达到此目的。如果无此语句，则 i 的值始终不改变，循环永远不结束。程序运行结果如图 5-3 所示。

4．使用 while 语句的注意事项

（1）循环体语句有可能一次也不执行。

（2）循环体可为任意类型语句。

（3）在循环体中，应用使循环趋于结束的语句，在上面例子中，i 的值大于 100 循环结束。如果没有使循环变量趋于终止的语句 i++，则循环变成无限循环（死循环）。

（4）死循环：while（1）循环体，即表达式的值为真的情况，可以在循环体中加入 break 语句来结束无限循环。

（5）循环体如果包含一个以上语句，应加花括号括起来，以复合语句的形式出现。否则，默认循环体语句到第一个分号处。

5.3　用 do…while 语句实现循环结构程序设计

1．一般形式

```
do
{循环体语句}
while(表达式);
```

2．执行过程

先执行循环体，然后再检查条件是否成立，若成立再执行循环体。这与 while 语句不同。执行过程如图 5-4 所示。

3．执行特点

先无条件地执行循环体，然后判断循环条件是否成立。

【例 5.2】用 do…while 语句实现例 5.1。

解题思路：与例 5.1 相似，用循环结构来处理，但题目要求用 do…while 语句来实现循环结构。先画出 N-S 图，如图 5-5 所示。

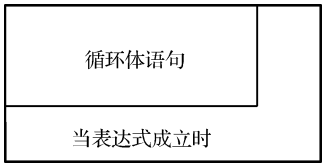


图 5-4　do…while 语句执行过程

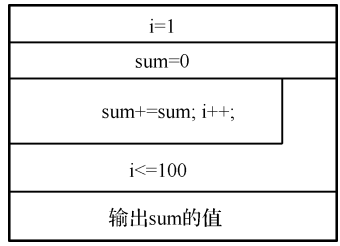


图 5-5　例 5.2 程序 N-S 图

```
#include<stdio.h>
int main()
{   int i=1,sum=0;
    do
    {sum=sum+i;
    i=i+1;
    } while(i<=100);
    printf(the result is "%d",sum);
    return 0;
}
```

```
the result is: 5050
```

图 5-6　例 5.2 程序运行结果

运行结果如图 5-6 所示。

程序分析：

（1）循环体放在 do 后的花括号里，多条语句用花括号括起来，作为复合语句出现。

（2）i 和 sum 赋初值应该在 do while 语句之前。

（3）在循环体中应有使循环趋向于结束的语句。本例中的 i++就是使循环能够结束的调整语句。程序运行结果如图 5-6 所示。

4．使用 do…while 语句的注意事项

（1）do…while 结构至少执行一次循环体，然后判断表达式。

（2）do…while 可以转换为 while 结构。

（3）对于同一个问题，既可以用 while 结构实现，也可以用 do…while 结构实现，二者在初值满足表达式时的结果值是相同的。但有时，它们的结果值却是不同的。因此，在处理每一个实际问题的时候，一定要注意 while 结果和 do…while 结构的不同之处，这样才能保证正确地使用两种结构。

（4）另外，do…while 语句也可以改写成 while 语句。

5.4 用 for 语句实现循环结构程序设计

1．一般形式

```
for(表达式 1；表达式 2；表达式 3)
    循环体语句
```

上述形式可解释为：

```
for(循环变量赋初值；循环条件；循环变量增值)
    循环体语句
```

2．执行过程

（1）求解表达式 1。

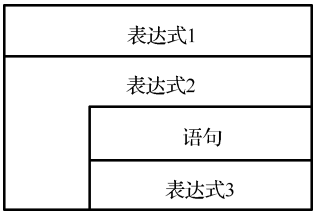


图 5-7 for 语句 N-S 图

（2）求解表达式 2，如果表达式 2 的值为非 0（真），则执行循环体语句后，执行下面的第（3）步。如果表达式 2 的值为 0（假），则转到第（4）步。

（3）求解表达式 3，然后转到（2）去执行。

（4）循环结束，执行 for 语句下面的语句。for 语句 N-S 图如图 5-7 所示。

3．执行特点

首先执行表达式 1，然后判断表达式 2，如果其为假，则一次循环体语句都不执行。

for 语句是 C 语言中使用最为灵活的语句，它不仅可以用于循环次数已经确定的情况下，而且循环次数未知时也可以使用。它可以完全取代 while 语句。并且，for 语句也可以改写成 while 语句的形式：

```
表达式 1;
while（表达式 2）
{循环体语句
 表达式 3}
```

因此，对于求出 1–100 的和可以用 for 语句实现，如下：

```
for(i=1;i<=100;i++)sum=sum+i;
```

【例 5.3】用 for 语句实现输出 1–100 之间的偶数，一行输出 10 个数，并求出偶数的个数。

解题思路：使用循环变量 i 来遍历 1～100，k 用来统计偶数个数。对于每一个 i，使用 if 语句判断其是否为偶数，如果是偶数则把 k 值加 1。这 100 次判断操作相同，可以用循环结构来处理。

```
#include<stdio.h>
int main()
{int i,k=0;
for(i=1;i<=100;i++)
    {if(i%2==0){printf("%5d",i);
                k=k+1;}
    if(k%10==0)printf("\n");
}
printf("%d\n",k);
return 0;}
```

```
   2    4    6    8   10   12   14   16   18   20
  22   24   26   28   30   32   34   36   38   40
  42   44   46   48   50   52   54   56   58   60
  62   64   66   68   70   72   74   76   78   80
  82   84   86   88   90   92   94   96   98  100
50
```

图 5-8　例 5.3 运行结果

运行结果如图 5-8 所示。

程序分析：使用循环来逐个判断 1～100 是否为偶数，每发现一个偶数，计数器 k 的值加 1，每次执行循环时，还要看 k 的值是否为 10 的倍数，若是则回车换行。程序运行结果如图 5-8 所示。

使用 for 语句的说明如下。

（1）for 语句中的“表达式 1”、“表达式 2”和“表达式 3”都是选择项，即可以缺省，但“;”不能缺省。

（2）省略了“表达式 1”，表示不对循环控制变量赋初值，此时应在 for 语句之前给循环变量赋初值。

（3）省略了“表达式 2”，则成为死循环，需做其他处理，对此将在后面介绍。

例如：

```
for(i=1;;i++)sum=sum+i;
```

相当于：

```
i=1;
while（1）
{sum=sum+i;
 i++;}
```

（4）省略了“表达式 3”，则不对循环控制变量进行操作，这时程序设计者应设法保证程序正常结束。

例如：

```
for(i=1;i<=100;)
{sum=sum+i;
      i++;}
```

（5）省略了“表达式 1”和“表达式 3”。

例如：

```
for(;i<=100;)
{sum=sum+i;
    i++;}
```

相当于：

```
while(i<=100)
```

```
{sum=sum+i;
 i++;}
```

但是，在两个循环结构的上面应有是循环变量赋初值的语句 i=1。

（6）三个表达式都可以省略。

例如：

```
for(  ;   ; )语句
```

相当于：

```
while（1）语句
```

在实际的使用过程中，应避免使用上面两种形式，因为如果程序中没有特殊的控制结构，它们就构成了死循环。

（7）表达式 1 可以是设置循环变量的初值的赋值表达式，也可以是其他表达式。

例如：

```
for(sum=0;i<=100;i++)sum=sum+i;
```

（8）表达式 1 和表达式 3 可以是一个简单表达式，也可以是逗号表达式。

```
for(sum=0,i=1;i<=100;i++)sum=sum+i;
```

或

```
for(i=0,j=10;i<=10&&j>5;i++,j--)k=i+j;
```

（9）表达式 2 一般是关系表达式或逻辑表达式，但也可是数值表达式或字符表达式，只要其值非零，就执行循环体。

例如：

```
for(i=0;(c=getchar())!='a';i++)putchar();
```

通过以上的学习，我们了解了 C 语言中实现循环结构的三种语句，并且知道 for 语句的应用最为灵活。循环语句也是我们以后在进行程序设计的过程中要频繁使用的一种语句，所以希望读者要好好掌握三种语句的使用规则。

5.5 循环的嵌套

一个循环体内又包含另一个完整的循环结构，称为循环的嵌套。内嵌的循环中还可以嵌套循环，构成多层嵌套。

三种循环可以相互嵌套。

说明：

（1）嵌套可以是多层的。

（2）一个循环体必须完完整整地嵌套在另一个循环体内，不能出现交叉。

（3）三种循环可以互相嵌套。

以下几种情况都是合法的嵌套结构。

（1）
```
while()
{ …
  while()
  { …
```

（2）
```
do
{ …
  do
  { …
```

```
        }                                    }while();
      …                                      …
    }                                      }while();
(3) while()                            (4) for( ; ;)
    { …                                    {…
    do                                       while()
    { …                                       {…
      }while();                                }
    …                                         …}
    }
(5) for(;;)                            (6) do
    {                                      {…
    for(;;)                                for(;;)
    {…}                                    {…}
    }                                      }while();
```

【例 5.4】使用嵌套语句输出金字塔形状。

解题思路：在本实例中，利用嵌套循环输出金字塔形状。显示一个三角形要考虑如下 3 点：（1）控制输出三角形的行数；（2）控制三角形的空白位置；（3）将三角形进行显示。

```
#include<stdio.h>
int main()
{
    int i, j, k;                          /*定义变量 i,j,k 为基本整型*/
    for (i = 1; i <= 5; i++)              /*控制行数*/
    {
        for (j = 1; j <= 5-i; j++)        /*控制空格数*/
            printf(" ");
        for (k = 1; k <= 2*i-1; k++)      /*控制打印*号的数量*/
            printf("*");
        printf("\n");
    }
     return 0;
}
```

图 5-9　例 5.4 运行结果

运行结果如图 5-9 所示。

程序分析：本程序包含一个双重循环，是 for 循环的嵌套。外层循环变量 i 由 1 变到 5，用来控制输出 5 行数据。内层循环变量 j，用来控制输出每行中空格的数量，内层循环变量 k 控制'*'的输出，注意每行最后输出一个'\n'，用来回车换行。程序运行结果如图 5-9 所示。

【例 5.5】循环嵌套的应用——输出九九乘法表。

解题思路：分析过程与上面的输出金字塔形状类似，首先注意控制行数，其次注意控制每行乘法算式个数和格式，最后还要注意换行。

编写程序：

```
#include<stdio.h>
void main()
{ int i,j;
 for(i=1;i<=9;i++)
```

```
    {for(j=1;j<=i;j++)
        printf("%d*%d=%2d",i,j,i*j);
        printf("\n");
     }
}
```

运行结果如图 5-10 所示。

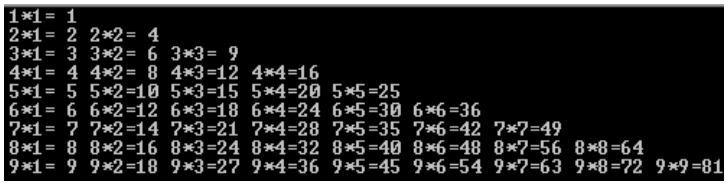


图 5-10　例 5.5 程序运行结果

程序分析：本程序包含一个双重循环，是 for 循环的嵌套。外层循环变量 i 由 1 变到 9，用来控制输出 9 行数据。内层循环变量 j 由 1 变到 i，用来控制输出每行中的 i 组数据。即每行输出数据的组数的最大值与行数是相同的，输出的形式为左下角三角形。输出的数据为 i，j，i*j。在执行第 1 次外循环体时，i=1，j=1，因此输出的结果为 1*1=1。在执行第 2 次外循环时，i=2，j 由 1 变化到 2，内层循环体执行 2 次，输出的结果分别为 2*1=2 和 2*2=4 两组数据。以下的情况以此类推。另外，内层循环中的 printf("\n")语句是用来控制输出换到的。程序运行结果如图 5-10 所示。若此处没有该语句，则输出的结果是对的，但是得不到如图 5-10 所示的这种显示形式。

5.6　几种循环的比较

前面介绍了三种可以执行循环操作的语句，这三种循环都可以解决同一个问题。

（1）三种循环一般可互相代替。

（2）while 和 do…while 循环的循环体中应包括使循环趋于结束的语句。

（3）for 循环可以在表达式 3 中包含使循环趋于结束的语句，甚至可以将循环体中的操作全部放到表达式 3 中。因此，for 语句的功能更强，完全可以代替 while 循环。

（4）用 while 和 do…while 循环时，循环变量初始化的操作应在 while 和 do…while 语句之前完成，而 for 语句可在表达式 1 中给出。

（5）while 和 for 循环是先判断表达式，后执行语句，而 do…while 循环是先执行语句，后判断表达式。

（6）对 while、do…while 和 for 循环，可以用 break 跳出循环，用 continue 语句结束本次循环。

5.7　break 和 continue 语句

5.7.1　break 语句

一般形式：break;

作用：

（1）跳出当前层循环，即提前结束循环，接着执行循环体下面的其他语句。

（2）break 语句只能用于循环语句和 switch 语句。

break 语句不能用于循环语句和 switch 语句之外的任何其他语句中。例如，在 while 循环语句中使用 break 语句：

```
while（1）
{   printf("break");
    break;
}
```

在程序中，虽然 while 语句是一个条件永远为真的循环，但是在其中使用 break 语句使得程序流程能够跳出循环。

注意：这个 break 语句和 switch…case 分支结构中的 break 语句的作用是不同的。

【例 5.6】使用 break 语句跳出循环。

解题思路：使用 for 语句执行循环输出 10 次的操作，在循环体中判断循环输出的次数。若为第 5 次输出，使用 break 语句跳出循环，终止循环输出操作。

```
#include<stdio.h>
int main()
{   int iCount;                          /*循环控制变量*/
    for(iCount=0;iCount<10;iCount++)     /*执行 10 次循环*/
    { if(iCount==5)                      /*判断条件，如果 iCount 等于 5 则跳出*/
        {printf("Break here\n");
         break;                          /*跳出循环*/ }
        printf("the counter is:%d\n",iCount);  /*输出循环的次数*/}
    return 0;
}
```

运行结果如图 5-11 所示。

```
the counter is:0
the counter is:1
the counter is:2
the counter is:3
the counter is:4
Break here
```

图 5-11　例 5.6 程序运行结果

程序分析：变量 iCount 在 for 语句中被赋值为 0，因为 iCount<10，所以循环执行 10 次。在循环语句中使用 if 语句判断当前 iCount 的值。当 iCount 值为 5 时，if 判断为真，使用 break 语句跳出循环。程序运行结果如图 5-11 所示。

5.7.2　continue 语句

在某些情况下，程序需要返回到循环头部继续执行，而不是跳出循环。continue 语句的一般形式是：

```
continue;
```

其作用就是结束本次循环，即跳过循环体中尚未执行的部分，接着执行下一次的循环操作。

【例 5.7】使用 continue 结束本次的循环操作。

解题思路：本实例与使用 break 语句结束循环的实例相似，区别在于将使用 break 语句的位置改写成了 continue。因为 continue 语句是结束一次循环，所以剩下的循环还是会继续执行。

```
#include<stdio.h>
int main( )
{   int iCount;                          /*循环控制变量*/
    for(iCount=0;iCount<10;iCount++)     /*执行 10 次循环*/
    {   if(iCount==5)                    /*判断条件，如果 iCount 等于则 5 跳出*/
        {   printf("Continue here\n");
```

```
            continue;                    /*跳出本次循环*/}
        printf("the counter is:%d\n",iCount);  /*输出循环的次数*/
    }
    return 0;
}
```

运行结果如图 5-12 所示。

程序分析：通过程序的显示结果，可以看到在 iCount 等于 5 时，调用 continue 语句使得本次的循环结束。但是，循环本身还没有结束，因此程序会继续执行。运行程序，显示效果如图 5-12 所示。

```
the counter is:0
the counter is:1
the counter is:2
the counter is:3
the counter is:4
Continue here
the counter is:6
the counter is:7
the counter is:8
the counter is:9
```

图 5-12　例 5.7 程序运行结果

5.7.3　break 和 continue 语句的区别

break 语句的作用是结束整个循环，不再判断执行循环的条件是否成立；continue 语句的作用从图 5-14 中可以看出，当表达式 2 为真的时候，所执行的 break 语句和 continue 语句的转向不同，即 break 语句终止 for 循环，而 continue 语句则是继续是否执行下一次循环的判断。break 语句和 continue 语句的流程图分别如图 5-13 和图 5-14 所示。

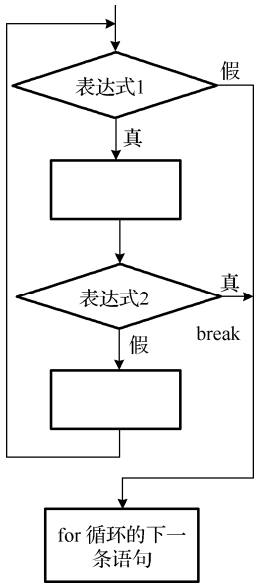


图 5-13　break 语句的流程图

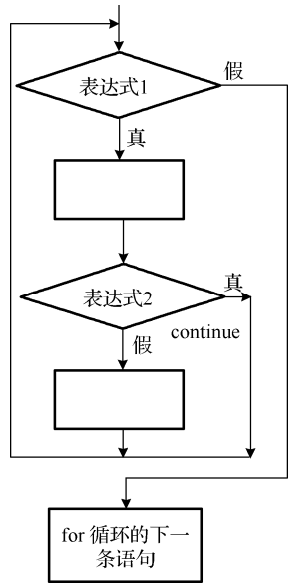


图 5-14　continue 语句的流程图

break 语句：

```
for(;表达式 1;)
{…
if(表达式 2)break;
  …
 }
```

continue 语句：

```
for(; 表达式 1;)
{…
   if(表达式 2)continue;
  …
}
```

5.8　程 序 举 例

【例 5.8】求 Fibonacci 数列的第 40 个数。该数列有这样的特点：第 1 个数和第 2 个数均为 1，

从第 3 个数开始，每个数都是前两个数的和。即

F1=1　　(n=1)

F2=1　　(n=2)

Fn= Fn–2+Fn–1　　(n≥3)

Fibonacci 数列是一个古典的数学问题：有一对兔子，从出生后第 3 个月起，每个月都生一对小兔子。小兔子长到第 3 个月后，每个月又生一对小兔子。假设所有兔子都不死，问题是：每个月的兔子总数是多少？

按照上面的说明，可以从表 5.1 中看出规律：不满一个月为小兔子，满一个月但不满两个月的为中兔子，超过三个月及以上的为老兔子。因此，可以看出兔子总数分别为：1，1，2，3，5，8，…，往下依此类推，这就是 Fibonacci 数列的特点，繁殖规律如表 5-1 所示。

解题思路：根据上面的说明，可以从前两个月的兔子数推出第 3 个月的兔子数。第 1、2 个月的兔子数分别是 f1=1，f2=1，则第 3 个月的兔子数 f3=f1+f2=2。往下继续类推，第 4 个月兔子数 f4=f2+f3=3，第 5 个月兔子数 f5=f3+f4=5，…，这样写下去，不但程序冗长烦琐，变量还需要多定义。因此，可以试想用同一变量来进行处理，即一个变量名在不同的月份代表不同的兔子数。

表 5-1　繁殖规律

第几个月	小兔子对数	中兔子对数	老兔子对数	兔子总数
1	1	0	0	1
2	0	1	0	1
3	1	0	1	2
4	1	1	1	3
5	2	1	2	5
6	3	2	3	8
…	…	…	…	…

说明：上表中不满一个月的为小兔子，满一个月但不满两个月的为中兔子，满三个月的为老兔子。

在开始时，分别设 f1、f2 分别代表第 1、2 个月的兔子数，初值均为 1，先把这两个月的兔子数输出。第 3 个月的兔子数为 f1+f2，但结果存放到哪呢？表达式 f1+f2 在参与运算时，f1 使用后就可以认为它是空闲的，因此可以把结果存放到 f1，所有第 3 个月兔子数为：f1(f3)=f1+f2。在计算完第 3 个月兔子数后，第 4 个月兔子数为 f2+f1，对此表达式，同上，因此可以写成 f2(f4)=f2+f1，以下的第 5 个月，第 6 个月，…，以此类推。算法用 N-S 图表示，如图 5-15 所示。

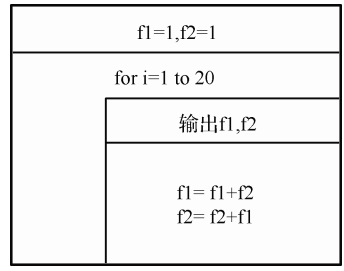


图 5-15　求 Fibonacci 数列的 N-S 图

编写程序：

```
#include<stdio.h>
void main()
{int i;
  long int f1,f2;
  f1=1;f2=1;
  for(i=1;i<=20;i++)          /*每循环一次，输出 2 个月的兔子数，故循环 20 次*/
  { printf("%12ld %12ld",f1,f2);    /* 输出已知的两个月兔子数*/
    if(i%2==0)printf("\n");
    f1= f1+f2;                /*计算下一个月的兔子数，存到 f1 中*/
    f2= f2+f1;                /*计算下两个月的兔子数，存到 f2 中*/
  }
}
```

运行结果如图 5-16 所示。

```
       1        1        2        3
       5        8       13       21
      34       55       89      144
     233      377      610      987
    1597     2584     4181     6765
   10946    17711    28657    46368
   75025   121393   196418   317811
  514229   832040  1346269  2178309
 3524578  5702887  9227465 14930352
24157817 39088169 63245986 102334155
```

图 5-16　例 5.9 程序运行结果

程序说明：程序要求输出到第 40 个数，即相当于输出第 40 个月的兔子数。因为所使用的编程环境为 Turbo C，因此变量 f1 和 f2 定义为 long int 型。输出函数 printf 中的格式说明符%12ld 为长整型的格式说明符。如果改为%12d，则输出出现错误。因为%md 的形式是整型数据的输入输出格式说明符，其最大值为 32767。运行结果如图 5-16 所示。

【例 5.9】从键盘上任意输入一个正整数，判断其是否为素数。

解题思路：素数是除了 1 和它本身之外没有任何其他约数的数。因此，对任意数 m，让 m 被 k 除（k 的值从 2 变化到 m–1），如果 m 能被 2～m–1 之中的任何一个数整除，则 m 一定不是素数。此时循环可以提前结束了。此时 k 的值一定是小于 m 的。可以用下面的程序段来说明：

```
int m,i;
scanf("%d",&m);
for(i=2;i<=m-1;i++)
if(m%i==0)break;
if(i==m)printf("%d is a prime\n",m);
else printf("%d is not a prime\n",m);
```

其实可以简化程序，即 m 不必被 2～m–1 范围内的各个整数去除。这里只需让 k 从 2 变化到 $\sqrt{m}$，后让 m 被 k 除，如果 m 能被 2～$\sqrt{m}$ 之中的任何一个整数整除，则循环结束，此时 i 必然小于或等于 k（即 $\sqrt{m}$），如果 m 不能被 2～$\sqrt{m}$ 之间的任一整数整除，则在完成最后一次循环后，i 还要加 1，因此 i=k+1，然后才终止循环。在循环之后判断 i 的值是否大于或等于 k+1，若条件满足，则表明未曾被 2～k 之间任一整数整除过，因此输出“是素数”。

编写程序：

```
#include<stdio.h>
#include<math.h>
void main()
{int i,k,m;
   scanf("%d",&m);
   k=sqrt(m);
   for(i=2;i<=k;i++)
   if(m%i==0)break;
if(i>=k+1)printf("%d is a prime number\n",m);
else printf("%d is not a prime number\n",m);
}
```

运行结果如图 5-17 所示。

```
19
19 is a prime number
```

图 5-17　例 5.9 程序运行结果

程序说明：如果 m 能被 2～k 之间的一个整数整除，此时只

需用 break 语句提前结束 for 循环，执行 for 语句下面的其他语句。问题的关键是：如何判断任意数 m 是否是素数？解决这个问题在于看结束循环时 i 的值与 k 值之间的关系。如果 m 能被 2～k 之间的任意一个数整除，则 i 一定小于或等于 k，否则循环正常结束，i 的值应该大于 k，即循环变量的值必然大于事先指定的循环变量终值。所以，只要在循环结束后检查循环变量 i 的值就可以判定循环是提前结束还是正常结束，也就可以判断出 m 是否是素数。程序运行结果如图 5-17 所示。

【例 5.10】输出 100～200 之间的所有素数。每行输出 10 个数。

解题思路：利用上面的分析过程，要求输出 100～200 之间的素数，就是求素数的过程作为循环体，利用循环嵌套的过程即可。

编写程序：

```
#include<stdio.h>
#include<math.h>
void main()
{
  int k,i,m,n=0;
for(m=101;m<=200;m=m+2)
  {
    k=sqrt(m);
    for(i=2;i<=k;i++)
      if(m%i==0)break;
    if(i>=k+1){printf("%d",m);n++;}
    if(n%5==0)printf("\n");
  }
  printf("\nn=%d\n",n);
}
```

```
101 103 107 109 113 127 131 137 139 149
151 157 163 167 173 179 181 191 193 197
199
n=21
```

图 5-18　例 5.10 程序运行结果

运行结果如图 5-18 所示。

程序说明：

（1）根据以前学过的知识，只有奇数才可能是素数，因此不必对偶数进行判断。故外层循环变量 m 从 101 开始，每次变化时增加 2。

（2）n 的作用是累加计数，确定素数的个数及用它来控制每行输出数据的个数。

（3）此程序循环体是双层循环，内层循环用来判断某个数是否是素数，外层循环用来控制内层循环的次数，更主要的是判断其中某些数是否是素数。并且在程序执行过程中，break 语句的功能只是跳出内层循环。程序运行结果如图 5-18 所示。

其实，对于求素数的过程还有下面的求法：标记变量法。它的基本思想是，首先假设所求的数据是素数，这时，设某一变量的值为 1（即为真值，例如 k=1），在求解的过程中，与上面的方法类似，即判断 m 是否是素数，可让 m 被 i 除（i 的值从 2 变化到 m-1），如果 m 能被 i 整除，此时置变量 k=0（即为假值），直到循环结束，k 一定有一个确定的值 1 或 0（即真或假）。该方法的程序段如下：

```
k=1;
for(i=2;i<=m-1;i++)
if(m%i==0)k=0;
if(k)printf("%d is a prime",m);
```

因此，上面的程序可以用下面的方法改写：

```
#include<stdio.h>
void main()
{
  int i,m,k,n=0;
  for(m=101;m<=200;m+=2)
  {
    k=1;
    for(i=2;i<m;i++)
    if(m%i==0)k=0;
    if(k)
    {   printf("%4d",m);
        n++;
        if(n%5==0)printf("\n");
    }
  }
    printf("\nn=%2d\n",n);
}
```

本 章 小 结

本章主要讲解了 C 语言中的一种重要的结构，即循环结构。C 语言提供了三种循环语句。

（1）for 语句主要用于给定了循环变量初值、有步长增量以及循环次数的循环结构。

（2）如果循环次数及控制条件要在循环过程中才能确定，则用 while 或 do…while 语句。

（3）三种循环语句可以相互嵌套组成多重循环。循环之间可以并列但不能交叉。

（4）可用转移语句把流程转出循环体外，但不能从外面转向循环体内。

（5）在循环程序中应避免出现死循环，即应保证循环变量的值在运行过程中可以得到修改，并使循环条件逐步变为假，从而结束循环。

习 题 5

一、选择题

1．while 循环语句中，while 后一对圆括号中表达式的值决定了循环体是否进行。因此，进入 while 循环后，一定有能使此表达式的值变为（　　）的操作，否则，循环将会无限制地进行下去。

A．0　　B．1　　C．成立　　D．2

2．在 do-while 循环中，循环由 do 开始，用 while 结束；必须注意的是：在 while 表达式后面的（　　）不能丢，它表示 do-while 语句的结束。

A．0　　B．1　　C．；　　D．，

3．for 语句中的表达式可以部分或全部省略，但两个（　　）不可省略。但当三个表达式均省略后，因缺少条件判断，循环会无限制地执行下去，形成死循环。

A．0　　B．1　　C．；　　D．，

4．程序段如下：

```
int k=-20;
while(k=0)  k=k+1;
```

说法正确的是（　　）。

A．while 循环执行 20 次　　B．循环是无限循环

C．循环体语句一次也不执行　　D．循环体语句执行一次

5．程序段如下：

```
int k=1;
while(!k==0)  {k=k+1;printf("%d\n",k);}
```

说法正确的是（　　）。

A．while 循环执行 2 次　　B．循环是无限循环

C．循环体语句一次也不执行　　D．循环体语句执行一次

二、填空题

1．while 语句的特点是_____，do…while 语句的特点是_____。

2．将 for(表达式 1;表达式 2;表达式 3)语句改写为 while 语句是_____。

3．break 语句的功能是_____。

4．break 语句只能用于_____语句和_____语句中。

5．continue 语句的作用是_____，即跳过循环体中下面尚未执行的语句，接着进行下一次是否执行循环的判定。

三、编程题

1．编写程序，求两个整数的最大公约数。

2．把输入的整数（最多不超过 5 位）按输入顺序的反方向输出。例如，输入数是 12345，要求输出结果是 54321，编程实现此功能。

3．中国古代数学家张丘建提出“百鸡问题”：一只大公鸡值五个钱，一只母鸡值三个钱，三个小鸡值一个钱。现在有 100 个钱，要买 100 只鸡，是否可以？若可以，给出一个解，要求三种鸡都有。请写出求解该问题的程序。

4．一个数如果恰好等于它的因子之和（除自身外），则称该数为完全数，例如：6=1+2+3，6 就是完全数，请编写一程序，求出 1000 以内的整数中的所有完全数。其中，1000 由用户输入。

5．编写一程序，求 1–3+5–7+…–99+101 的值。

第6章　数　　组

6.1　为什么使用数组

整型、浮点型、字符型是C语言提供的基本数据类型。在编写程序的过程中，经常遇到使用很多数据量的情况，处理每一个数据量都要有一个相对应的变量。如果每一个变量都要单独进行定义，则会很烦琐，使用数组可以解决这种问题。而在程序设计中，为了处理方便，把具有相同数据类型的若干变量按有序的形式组织起来。这些按序排列的同类数元素的集合称为数组。在C语言中，数组属于构造数据类型（除了指针类型外，复杂数据类型又称为构造类型，是由基本数据类型组合而成的）。本章介绍数组的定义、引用、初始化方式及数组的应用。

6.2　一 维 数 组

6.2.1　一维数组的定义

从概念上来说，数组是一组变量，这组变量应该满足下列条件：

（1）具有相同的名字。

（2）具有相同的数据类型。

（3）在存储器中连续存放。

其中，每个变量称为数组的一个“数据单元”，保存在其中的数据值称为“数组元素”。数组对象的整体有一个名称，这个名称表示整个数组。

在C语言中使用数组必须先进行定义。

一维数组的定义方式为：

数据类型说明符 数组名 [常量表达式]；

其中，数据类型说明符是任一种基本数据类型或构造数据类型。数组名是用户定义的数组标识符。方括号中的常量表达式表示数据元素的个数，也称为数组的长度。

例如：

```
int a[10];              说明整型数组 a，有 10 个元素。
float b[10],c[20];      说明实型数组 b，有 10 个元素，实型数组 c，有 20 个元素。
char string[20];        说明字符数组 string，有 20 个元素。
```

数组定义时应注意以下几点：

（1）数组名的书写规则应符合标识符的书写规定。

（2）数组的数据类型实际上是指数组元素的取值类型。对于同一个数组，其所有元素的数据类型都是相同的。

（3）数组名不能与其他变量名相同。

例如：

```
main()
```

```
{
    int a;
    float a[10];     /*与变量 i 同名*/
    …}
```

是错误的。方括号中常量表达式表示数组元素的个数，如 a[5]表示数组 a 有 5 个元素。但是其下标从 0 开始计算。因此，5 个元素分别为 a[0],a[1],a[2],a[3],a[4]。

（4）不能在方括号中用变量来表示元素的个数，但是可以是符号常数或常量表达式。

例如：

```
#define FD 5
main()
{
    int a[3+2],b[7+FD];
    …
}
```

是合法的。

但是，下述说明方式是错误的。

```
main()
{
     int n=5;
     int a[n];
     …
}
```

（5）允许在同一个类型说明中，说明多个数组和多个变量。

例如：

```
int a,b,c,d,k1[10],k2[20];
```

定义好一个数组后，C 语言编译器会为其分配一片连续的内存空间，空间的大小由数组元素的个数和类型决定，即数组元素的个数×每个元素所占空间长度，可使用操作符 sizeof（数组名）来计算数组所占的内存空间。如 int a[3]有 3 个整型元素，每个元素所占空间为 4 个字节，则数组 a 所占空间为：3×4 字节=12 字节，即 sizeof(a)=12。数组 a 在内存中的存储形式如图 6-1 所示。

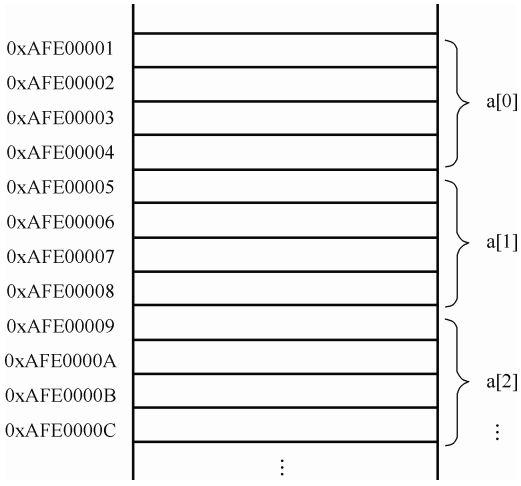


图 6-1　数组内存存储方式

6.2.2 一维数组的引用

数组元素是组成数组的基本单元。数组元素也是一种变量，其标识方法为数组名后跟一个下标。下标表示了元素在数组中的顺序号。

数组元素的一般形式为：

数组名[下标]

其中，下标只能为整型常量或整型表达式。如为小数时，C 编译将自动取整。

例如：

```
a[5]
a[i+j]
a[i++]
```

都是合法的数组元素。

数组元素通常也称为下标变量。必须先定义数组， 才能使用下标变量。在 C 语言中只能逐个地使用下标变量，而不能一次引用整个数组。

例如，输出有 10 个元素的数组必须使用循环语句逐个输出各下标变量：

```
for(i=0; i<10; i++)
    printf("%d",a[i]);
```

而不能用一个语句输出整个数组。

下面的写法是错误的：

```
printf("%d",a);
```

【例 6.1】定义一个整型数组存储 0～9，并输出这 10 个整数。

解题思路：为了保存 10 个整型数据，需要定义一个长度为 10 的数组，每个数组元素保存一个数据。由于数据值没有超出 int 类型范围，所以可将此数组定义成 int 类型。

```
#define SIZE 10
main()
{
  int i,a[SIZE];
  for(i=0;i< SIZE;i++)
     a[i]=i;
  for(i= 0;i<SIZE;i++)
     printf("%d ",a[i]);
}
```

图 6-2 例 6.1 运行结果

运行结果如图 6-2 所示。

程序分析：语句#define SIZE 10 定义了符号常量 SIZE，用于代表数组长度，这是 C 语言一种常用的编程技巧。如果数组长度发生变化，只要改变 SIZE 的定义即可，不必改动程序中涉及数组长度的地方，增加了程序的可维护性。

a[i]是对数组元素的引用，i 为变量，但在每次循环中，i 都有确定的值，如第一次循环时，i 值为 0，a[i]=i 等价于 a[0]=0，第二次循环时 a[i]=i 等价于 a[1]=1，其他依此类推。显示效果如图 6-2 所示。

【例 6.2】在本例中，使用数组保存用户输入的数据，当输入完毕后逆向输出数据。

```
#include<stdio.h>
```

```
int main()
{   int iArray[5], index, temp;              /*定义数组及变量为基本整型*/
    printf("Please enter a Array:\n");
    for (index= 0; index< 5; index++)        /*逐个输入数组元素*/
    {
        scanf("%d", &iArray[index]);
    }
    printf("Original Array is:\n");
    for (index = 0; index< 5; index++)       /*显示数组中的元素*/
    {
        printf("%d", iArray[index]);
    }
    printf("\n");

    for (index= 0; index < 2; index++)       /*将数组中元素的前后位置互换*/
    {
       temp = iArray[index];          /*元素位置互换的过程借助中间变量 temp*/
       iArray[index] = iArray[4-index];
       iArray[4-index] = temp;
    }
    printf("Now Array is:\n");
    for (index = 0; index< 5; index++)       /*将转换后的数组再次输出*/
    {
        printf("%d", iArray[index]);
    }
    printf("\n");
    return 0;
}
```

运行结果如图 6-3 所示。

```
Please enter a Array:
11
22
33
44
55
Original Array is:
11 22 33 44 55
Now Array is:
55 44 33 22 11
Press any key to continue
```

图 6-3　例 6.2 运行结果

分析：在本实例中，程序定义变量 temp 用来实现数据间的转换，而 index 用于控制循环的变量。通过语句 int iArray[5]定义一个有 5 个元素的数组，程序中用到的 iArray[i]就是对数组元素的引用。运行程序，显示效果如图 6-3 所示。

6.2.3　一维数组的初始化

给数组赋值的方法除了用赋值语句对数组元素逐个赋值外，还可采用初始化赋值和动态赋值的方法。

数组初始化赋值是指在数组定义时给数组元素赋予初值。数组初始化是在编译阶段进行的。这样将减少运行时间，提高效率。

初始化赋值的一般形式为：

类型说明符 数组名[常量表达式]={值，值，…，值}；

其中，在{ }中的各数据值即各元素的初值，各值之间用逗号间隔。

例如：

```
int a[10]={ 0,1,2,3,4,5,6,7,8,9 };
```

相当于 a[0]=0;a[1]=1...a[9]=9;

C 语言对数组的初始化赋值还有以下几点规定。

（1）可以只给部分元素赋初值。

当{ }中值的个数少于元素个数时，只给前面部分元素赋值。

例如：

```
int a[10]={0,1,2,3,4};
```

表示只给 a[0]～a[4]5 个元素赋值，而后 5 个元素自动赋 0 值。

（2）只能给元素逐个赋值，不能给数组整体赋值。

例如给 10 个元素全部赋 1 值，只能写为：

```
int a[10]={1,1,1,1,1,1,1,1,1,1};
```

而不能写为：

```
int a[10]=1;
```

（3）如给全部元素赋值，则在数组说明中，可以不给出数组元素的个数。

例如：

```
int a[5]={1,2,3,4,5};
```

可写为：

```
int a[]={1,2,3,4,5};
```

6.2.4 一维数组程序设计举例

【例 6.3】用数组来处理求 Fibonacci 数列问题。

解题思路：在第 5 章例 5.9 中是用简单变量处理的，只定义了 2 个或 3 个变量，程序可以顺序计算并输出各数，但不能在内存中保存这些数。假如想直接输出数列中第 25 个数，是很困难的。如果用数组来处理，在概念上反而简单了：每一个数组元素代表数列中的一个数，依次求出各数并存放在相应的数组元素中即可。

编写程序：

```
#include <stdio.h>
int main()
 {
  int i;
  int f[20]={1,1};                //对最前面两个元素 f[0]和 f[1]赋初值 1
  for(i=2;i<20;i++)
    f[i]=f[i-2]+f[i-1];           //先后求出 f[2]～f[19]的值
  for(i=0;i<20;i++)
    {
     if(i%5==0) printf("\n");     //控制每输出 5 个数后换行
       printf("%12d",f[i]);
    }
  printf("\n");
  return 0;
}
```

运行结果如图 6-4 所示。

```
  1         1         2         3         5
  8        13        21        34        55
 89       144       233       377       610
987      1597      2584      4181      6765
```

图 6-4　例 6.3 运行结果

程序分析：根据数列的特点，由前面 2 个元素的值可计算出第 3 个元素的值，f[2]=f[0]+f[1]；在循环中可以用以下语句依次计算出 f[2]～f[19]的值。

```
f[i]=f[i-2]+f[i-1];
```

if 语句用来控制换行，每行输出 5 个数据。显示效果如图 6-4 所示。

【例 6.4】输入 10 个整数，输出其中最大者。

解题思路：本题采用擂台算法，首先定义一个长度为 10 的整型数组，输入 10 个整数保存在数组中。再定义一个辅助变量用于保存最大值，把数组的第 1 个元素赋值给辅助变量。让辅助变量与第 2 个元素比较，若第 2 个元素大，则将值赋给辅助变量，否则不变。然后按同样的方法让辅助变量依次与数组其他各个元素进行比较，当整个数组比较完后，辅助变量中保存的就是数组中的最大值。程序 N-S 图如图 6-5 所示。

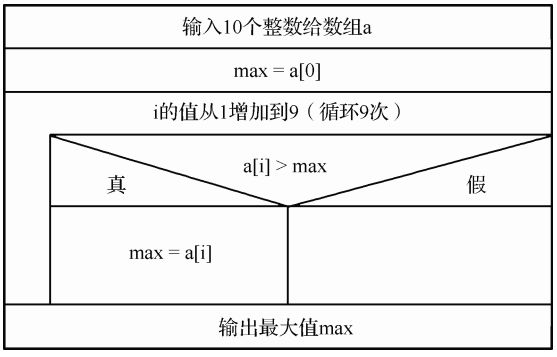


图 6-5　例 6.4 程序 N-S 图

编写程序：

```
#include <stdio.h>
#define SIZE 10
int main(void)
{   int i, a[SIZE];
    int max;
    for(i=0; i< SIZE; i++)
    {
        printf("\n 请输入第%d 个整数:", i+1);
        scanf("%d", &a[i]);
    }
    max = a[0];
    for(i=1; i< SIZE; i++)
    {
        if(a[i]>max)    //如果数组元素大，则与 max 互换
            max = a[i];
    }
```

```
        printf("最大值是:%d\n",max);
        return 0;
    }
```

运行结果如图 6-6 所示。

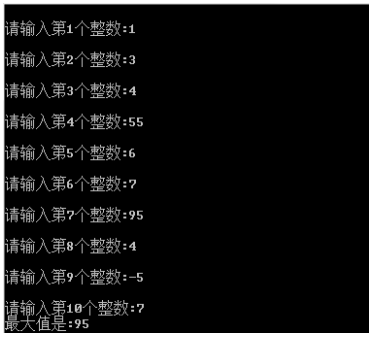


图 6-6　例 6.5 运行结果

程序分析：本例题通过数组元素的两两比较，取其大者，再依次往下比较来找出最大者，为了方便编程，增加了一个辅助变量 max 用于比较。程序主要由两个循环构成，第 1 个循环给数组输入 10 个整数，第 2 个循环用于求最大值，从 i=1（第 2 个数组元素）开始循环与 max 比较，将大者存入 max 中。显示效果如图 6-6 所示。

【例 6.5】使用冒泡排序法对这 10 个数由大到小排序。

排序是程序设计中非常重要的内容，它的功能是将一组无序的数据排列成有序的数据序列。一般情况下有升序和降序两种排列方式。

例如，我们统计班级学生的成绩，如果按照学号来进行统计，成绩是无序排列的，这样不利于我们对成绩的查询。因此在进行成绩查询之前，一般要先进行排序，如按照高分到低分的排序，这样就可以很快地查出本班的最高分和最低分，以及成绩比较靠前或靠后的学生。

冒泡排序的基本思想：对于 n 个数进行排序（现假定是从大到小排序，以下均按此进行），将相邻两个数依次比较，将大数调在前头：也就是说第 1 个数和第 2 个数比较，大数放前，小数放后，第 2 个和第 3 个进行比较，大数放前，小数放后，然后依此类推……经过第一轮比较以后，我们找到一个最小数在最下面（沉底）。然后进行下一轮比较，最后一个数就不用再参加比较了，所以本轮就可以少比较一次。

很显然，需要用双重循环来设计这个问题，外层循环控制进行的轮数，内层循环控制每轮比较的次数，那么到底需要多少轮、每轮需要多少次？下面通过一个实例加以说明。

排序过程举例（对 5 个数进行从大到小的排序）：过程如表 6-1 所示。

表 6-1　冒泡法排序表

外循环	1 轮				2 轮			3 轮		4 轮
内循环	5 个数比较 4 次				4 个数比较 3 次			3 个数比较 2 次		2 个数比较 1 次
	1 次	2 次	3 次	4 次	1 次	2 次	3 次	1 次	2 次	1 次
7	7	7	7	7	8	8	8	8	8	9
5	5	8	8	8	7	7	7	7	9	8
8	8	5	6	6	6	6	9	9	7	7
6	6	6	5	9	9	9	6	6	6	6
9	9	9	9	5	5	5	5	5	5	5
	最小的数 5 沉底，其余 4 个数继续比较				次小数 6 沉底，其余 3 个数			7 沉底，其余 2 个数比较		最后两个数一次比较

通过这个排序过程，我们了解了怎样去进行排序。那么，到底什么是气泡呢？我们可以从中找出答案，从大到小进行排序，较大的一些数就是气泡。随着排序的进行，气泡逐步上升。

从这个排序过程中还可以看出，n 个数最多需要 n–1 轮排序就可以把顺序确定出来，并且如果我们用 j 来表示轮次的话，那么每一轮比较的次数为 n–j。

因此，我们可以得到冒泡排序法的算法（对 5 个数进行从大到小的排序）。

```
for(i=0;i<5;i++)
   for(j=0;j<5-i;j++)
      if(a[j]<a[j+1])
        {t=a[j];a[j]=a[j+1];a[j+1]=t;}
```

10 个数冒泡法排序程序如下：

```
#include <stdio.h>
int main()
 {
   int a[10];
   int i,j,t;
   printf("input 10 numbers:\n");
   for (i=0;i<10;i++)
     scanf("%d",&a[i]);
   printf("\n");
   for(j=0;j<9;j++)                    //进行 9 次循环，实现 9 趟比较
    for(i=0;i<9-j;i++)                 //在每一趟中进行 9-j 次比较
      if (a[i]>a[i+1])                 //相邻两个数比较
        {t=a[i];a[i]=a[i+1];a[i+1]=t;}
   printf("the sorted numbers:\n");
   for(i=0;i<10;i++)
     printf("%d",a[i]);
   printf("\n");
   return 0;
}
```

```
input 10 numbers :
1 33 5 66 34 6 -9 4 7 20

the sorted numbers :
-9 1 4 5 6 7 20 33 34 66
```

运行结果如图 6-7 所示。

图 6-7　例 6.5 运行结果

程序分析：程序中实现冒泡法排序算法的主要是 10～13 行。请仔细分析嵌套的 for 语句。当执行外循环第 1 次循环时，j=0，然后执行第 1 次内循环，此时 i=0，在 if 语句中将 a[i]和 a[i+1]比较，就是将 a[0]和 a[l]比较。执行第 2 次内循环时，i=1，a[i]和 a[i+1]比较，就是将 a[l]和 a[2]比较……执行最后一次内循环时，i=8，a[i]和 a[i+1]比较，就是将 a[8]和 a[9]比较。这时第 1 趟过程完成了。

当执行第 2 次外循环时，j=1，开始第 2 趟过程。内循环继续的条件是 i<9-j，由于 j=1，因此相当于 i<8，即 i 由 0 变到 7，要执行内循环 8 次。其余类推。显示效果如图 6-7 所示。

6.3　二 维 数 组

6.3.1　二维数组的定义

前面介绍的数组只有一个下标，称为一维数组，其数组元素也称为单下标变量。在实际问题

中有很多量是二维的或多维的，因此C语言允许构造多维数组。多维数组元素有多个下标，以标识它在数组中的位置，所以也称为多下标变量。本节只介绍二维数组，多维数组可由二维数组类推而得到。

二维数组定义的一般形式是：

类型说明符 数组名[常量表达式 1][常量表达式 2]

（1）其中，常量表达式 1 表示第一维下标的长度，常量表达式 2 表示第二维下标的长度。

例如：

```
int a[3][4];
```

说明了一个 3 行 4 列的数组，数组名为 a，其下标变量的类型为整型。该数组的下标变量共有 3×4 个，即

```
a[0][0],a[0][1],a[0][2],a[0][3]
a[1][0],a[1][1],a[1][2],a[1][3]
a[2][0],a[2][1],a[2][2],a[2][3]
```

（2）二维数组在概念上是二维的，即其下标在两个方向上变化，下标变量在数组中的位置也处于一个平面之中，而不是像一维数组只是一个向量。但是，实际的硬件存储器却是连续编址的，即存储器单元是按一维线性排列的。在一维存储器中存放二维数组可有两种方式：一种是按行排列，即放完一行之后顺次放入第 2 行；另一种是按列排列，即放完一列之后再顺次放入第 2 列。在C语言中，二维数组是按行排列的。即先存放 a[0]行，再存放 a[1]行，最后存放 a[2]行。每行中有 4 个元素也是依次存放的。由于数组 a 说明为 int 类型，该类型占两个字节的内存空间，所以每个元素均占有两个字节。

例如，a[3][2]在内存中的存放顺序如图 6-8 所示。

a[0][0]
a[0][1]
a[1][0]
a[1][1]
a[2][0]
a[2][1]

图 6-8 二维数组存放顺序

6.3.2 二维数组的引用

二维数组的元素也称为双下标变量，其表示的形式为：

数组名[下标][下标]

其中，下标应为整型常量或整型表达式。

注意：下标变量和数组说明在形式中有些相似，但这两者具有完全不同的含义。数组说明的方括号中给出的是某一维的长度，即可取下标的最大值；而数组元素中的下标是该元素在数组中的位置标识。前者只能是常量，后者可以是常量、变量或表达式。

在引用数组元素时，下标值应在已定义的数组大小的范围内。在这问题上常出现错误。例如：

```
int a[3][4];          //定义 a 为 3×4 的二维数组
…
a[3][4]=3;            //错误，不存在 a[3][4]元素
```

按以上的定义，数组 a 可用的“行下标”的范围为 0～2，“列下标”的范围为 0～3。用 a[3][4]表示元素显然超过了数组的范围。

请读者严格区分在定义数组时用的 a[3][4]和引用元素时的 a[3][4]的区别。前者用 a[3][4]来定义数组的维数和各维的大小，后者 a[3][4]中的 3 和 4 是数组元素的下标值，a[3][4]代表行序号为 3、列序号为 4 的元素（行序号和列序号均从 0 起算）。

【例 6.6】任意输入一个 3 行 3 列的二维数组，求对角之和。

解题思路：在本例中，可以使用二维数组保存一个 3 行 3 列的数组，利用双重循环访问数组中的每一个元素。在循环中判断是否是对角线上的元素，对角线元素满足行标和列标相等的条件，然后进行对角线元素累加计算。

```
#include<stdio.h>
int main()
{
    int a[3][3];                          /*定义一个 3 行 3 列的数组*/
    int i,j,sum=0;                        /*定义循环控制变量和保存数据变量 sum*/
    printf("please input:\n");
    for(i=0;i<3;i++)                      /*利用循环进行对数组元素进行赋值*/
    {
        for(j=0;j<3;j++)
        scanf("%d",&a[i][j]);
    }
    for(i=0;i<3;i++)                      /*使用循环进行计算对角线的总和*/
    {
        for(j=0;j<3;j++)
            if(i==j) sum=sum+a[i][j];     /*进行数据的累加计算*/
    }
    printf("the result is:%d\n",sum);     /*输出最后的结果*/
    return 0;
}
```

运行结果如图 6-9 所示。

```
please input:
1 2 3
4 5 6
7 8 9
the result is :15
```

图 6-9　例 6.6 运行结果

程序分析：本例用了两个双重循环，第一个双重 for 循环用来对数组元素进行赋值，再利用第二个双重循环访问数组中的每一个元素，查找对角线上的元素，并进行累加计算。显示效果如图 6-9 所示。

6.3.3　二维数组的初始化

二维数组初始化也是在类型说明时给各下标变量赋以初值。二维数组可按行分段赋值，也可按行连续赋值。

例如对数组 a[5][3]：

（1）按行分段赋值可写为：

```
int a[5][3]={ {80,75,92},{61,65,71},{59,63,70},{85,87,90},{76,77,85} };
```

（2）按行连续赋值可写为:

```
int a[5][3]={ 80,75,92,61,65,71,59,63,70,85,87,90,76,77,85};
```

这两种赋初值的结果是完全相同的。

对于二维数组初始化赋值还有以下说明：

（3）可以只对部分元素赋初值，未赋初值的元素自动取 0 值。

例如：

```
int a[3][3]={{1},{2},{3}};
```

是对每一行的第一列元素赋值，未赋值的元素取 0 值。 赋值后各元素的值为：

```
1 0 0
```

```
2 0 0
3 0 0
int a [3][3]={{0,1},{0,0,2},{3}};
```

赋值后的元素值为:

```
0 1 0
0 0 2
3 0 0
```

（4）如对全部元素赋初值，则第一维的长度可以不给出。

例如：

```
int a[3][3]={1,2,3,4,5,6,7,8,9};
```

可以写为：

```
int a[][3]={1,2,3,4,5,6,7,8,9};
```

（5）数组是一种构造类型的数据。二维数组可以看成由一维数组的嵌套而构成的。设一维数组的每个元素都又是一个数组，就组成了二维数组。当然，前提是各元素类型必须相同。根据这样的分析，一个二维数组也可以分解为多个一维数组。C 语言允许这种分解。

如二维数组 a[3][4]，可分解为三个一维数组，其数组名分别为：

```
a[0]
a[1]
a[2]
```

对这三个一维数组不需另做说明即可使用。这三个一维数组都有 4 个元素，例如：一维数组 a[0]的元素为 a[0][0],a[0][1],a[0][2],a[0][3]。必须强调的是，a[0],a[1],a[2]不能作为下标变量使用，它们是数组名，不是一个单纯的下标变量。

6.3.4 二维数组程序设计举例

【例 6.7】将以下二维数组行和列元素互换，放到另个二维数组中。

$$a=\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{pmatrix} \qquad b=\begin{pmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{pmatrix}$$

解题思路：数组 a 是 2 行 3 列的矩阵，互换后，数组 b 应定义为 3 行 2 列矩阵，二维数组行和列元素互换的关键是元素坐标的调整，即 b[j][i]=a[i][j]。

程序如下：

```
#include<stdio.h>
main()
{
   int a[2][3]={{1,2,3},{4,5,6}};
   int b[3][2], i, j;
   printf("array a:\n");
   for(i=0;i<2;i++)              //输出转置前的矩阵
    {
      for(j=0;j<3;j++)
        printf("%5d",a[i][j]);
```

```
        printf("\n");
      }
    for(i=0;i<2;i++)
      for(j=0;j<3;j++)
        b[j][i]=a[i][j];          //行列元素互换
    printf("array b:\n");
    for(i=0;i<3;i++)
      {
        for(j=0;j<2;j++)
          printf("%5d",b[i][j]);//输出互换后的矩阵
        printf("\n");
      }
}
```

```
array a:
    1    2    3
    4    5    6
array b:
    1    4
    2    5
    3    6
```

图 6-10　例 6.7 运行结果

运行结果如图 6-10 所示。

程序分析：本题用了三次双重 for 循环，第一个双重循环用于输出转换前的矩阵，第二个双重循环用来行列互换，第三个双重循环用于输出转换后的新矩阵。显示效果如图 6-10 所示。

【例 6.8】有一个 3×4 的矩阵，编程求出其中值最大的那个元素的值，以及其所在的行号和列号。

解题思路：本题和例 6.6 类似，都采用擂台算法，使用双重循环遍历数组所有元素，找到最大值。

程序如下：

```
#include<stdio.h>
main()
{
int i,j,row=0,colum=0,max;
   int a[3][4]={{1,2,3,4},{9,8,7,10},{-10,6,-5,2}};
   max=a[0][0]; i=j=0;
   for(i=0;i<=2;i++)
     for(j=0;j<=3;j++)
       if(a[i][j]>max)
        {max=a[i][j];row=i;colum=j;}
   printf("max=%d,row=%d,colum=%d\n",max,row,colum);
}
```

```
max=10,row=1,colum=3
```

图 6-11　例 6.8 运行结果

运行结果如图 6-11 所示。

分析：变量 max 就是“擂台”变量，在遍历数组过程中若发现更大元素，就更新擂台 max 中的值，并更新最大值元素的下标 i 和 j。显示效果如图 6-11 所示。

6.4　字 符 数 组

数组中的元素类型为字符型时称为字符数组。字符数组中的每一个元素可以存放一个字符。字符数组的定义和使用方法与其他基本类型的数组基本相似。

6.4.1　字符数组的定义

字符数组的定义与其他数据类型的数组定义类似，一般形式如下。

一维字符数组定义的语法为：

char 数组名[常量表达式]；

例如：char name[10];

其中的 name 表示数组名，而括号中的 10 表示数组中包含 10 个字符型的变量元素。

二维字符数组定义的语法为：

char 数组名[常量表达式 1] [常量表达式 2]；

因为要定义的是字符型数据，所以在数组标识符前所用的类型是 char 后面括号中表示的是数组元素的数量。

6.4.2 字符数组的初始化

字符数组也允许在定义时做初始化赋值。

例如：

```
char c[10]={'c',' ','p','r','o','g','r','a','m'};
```

赋值后各元素的值为：

```
数组 c    c[0]的值为'c'
          c[1]的值为' '
          c[2]的值为'p'
          c[3]的值为'r'
          c[4]的值为'0'
          c[5]的值为'g'
          c[6]的值为'r'
          c[7]的值为'a'
          c[8]的值为'm'
```

其中，c[9]未赋值，系统自动赋予空字符（空字符是 ASCII 值为 0 的字符，空字符无法从键盘输入，所以用转义字符表示成'\0'）。

当对全体元素赋初值时也可以省去长度说明。

例如：

```
char c[]={'c',' ','p','r','o','g','r','a','m'};
```

这时 C 数组的长度自动定为 9。

6.4.3 字符数组的引用

字符数组的引用和其他类型数据引用一样，也是使用下标的形式。例如：

```
char c[10];
```

字符数组也可以是二维或多维数组。

例如：

```
char c[5][10];
```

即为二维字符数组。

【例 6.9】使用字符数组输出一个字符串。

思路分析：在本实例中，定义一个字符数组，通过初始化操作保存一个字符串，然后通过循环引用每一个数组元素进行输出操作。

```
#include<stdio.h>
int main()
{   char cArray[5]={'H','e','l','l','o'};  /*初始化字符数组*/
    int i;                                 /*循环控制变量*/
    for(i=0;i<5;i++)                       /*进行循环*/
    {   printf("%c",cArray[i]);            /*输出字符数组元素*/}
    printf("\n");                          /*输出换行*/
    return 0;
}
```

```
Hello
```

图 6-12　例 6.9 运行结果

运行结果如图 6-12 所示。

程序分析：在初始化字符数组时要注意，每一个元素的字符都是使用一对单引号进行表示的。在循环中，因为输出的类型是字符型，所以在 printf 中使用的是%c。通过循环变量 i，cArray[i]是对数组中每一个元素的引用。运行程序，显示效果如图 6-12 所示。

6.4.4　字符串和字符串结束标志

在 C 语言中没有专门的字符串变量，通常用一个字符数组来存放一个字符串。前面介绍字符串常量时，已说明字符串总是以'\0'作为串的结束符。因此当把一个字符串存入一个数组时，也把结束符'\0'存入数组，并以此作为该字符串是否结束的标志。有了'\0'标志后，就不必再用字符数组的长度来判断字符串的长度了。

C 语言允许用字符串的方式对数组做初始化赋值。

例如：

```
char c[]={'c',' ','p','r','o','g','r','a','m'};
```

可写为：

```
char c[]={"C program"};
```

或去掉{}写为：

```
char c[]="program";
```

用字符串方式赋值比用字符逐个赋值要多占一个字节，用于存放字符串结束标志'\0'。上面的数组 c 在内存中的实际存放情况为：

C		p	r	o	g	R	a	m	\0

'\0'是由 C 编译系统自动加上的。由于采用了'\0'标志，所以在用字符串赋初值时一般无须指定数组的长度，而由系统自行处理。

6.4.5　字符数组的输入/输出

在采用字符串方式后，字符数组的输入/输出将变得简单方便。

除了上述用字符串赋初值的办法外，还可用 printf()函数和 scanf()函数一次性地输入/输出一个字符数组中的字符串，而不必使用循环语句逐个地输入/输出每个字符。

【例 6.10】在本实例中为定义的字符数组进行初始化操作，再输出字符数组中保存的数据时，可以逐个将数组中的元素进行输出，或者直接将字符串进行输出。

```
#include<stdio.h>
int main()
{   int iIndex;                            /*循环控制变量*/
```

```
    char cArray[12]="MingRi KeJi";      /*定义字符数组用于保存字符串*/
    for(iIndex=0;iIndex<12;iIndex++)
    {   printf("%c",cArray[iIndex]);    /*逐个输出字符数组中的字符*/    }
    printf("\n%s\n",cArray);            /*直接将字符串输出*/
    return 0;
}
```

运行结果如图 6-13 所示。

程序分析：在代码中，对数组中元素逐个输出时使用的是循环的方式，而直接输出字符串是利用 printf 函数中的格式符"%s"进行输出。要注意，直接输出字符串时不能使用格式符"%c"。运行程序，显示效果如图 6-13 所示。

【例 6.11】

```
main()
{
   char st[15];
   printf("input string:\n");
   scanf("%s",st);
   printf("%s\n",st);
}
```

运行结果如图 6-14 所示。

程序分析：本例中由于定义数组长度为 15，因此输入的字符串长度必须小于 15，以留出一个字节用于存放字符串结束标志'\0'。应该说明的是，对一个字符数组，如果不做初始化赋值，则必须说明数组长度。还应该特别注意的是，当用 scanf()函数输入字符串时，字符串中不能含有空格，否则将以空格作为串的结束符。显示效果如图 6-14 所示。

例如，当输入的字符串中含有空格时，运行结果如图 6-15 所示。

```
MingRi KeJi
MingRi KeJi
```

图 6-13　例 6.10 运行结果

```
input string:
abcabcabc
abcabcabc
```

图 6-14　例 6.11 运行结果一

```
input string:
This is a cat
This
```

图 6-15　例 6.11 运行结果二

从输出结果可以看出空格以后的字符都未能输出，为了避免这种情况，可多设几个字符数组分段存放含空格的串。

6.4.6　字符串处理函数

C 语言提供了丰富的字符串处理函数，大致可分为字符串的输入、输出、合并、修改、比较、转换、复制、搜索几类。使用这些函数可大大减轻编程的负担。用于输入/输出的字符串函数，在使用前应包含头文件"stdio.h"，使用其他字符串函数则应包含头文件"string.h"。

下面介绍几个最常用的字符串函数。

1．字符串输出函数 puts()

格式：puts(字符数组名)

功能：puts()函数使用很简单，只需要给出保存字符串的字符数组名即可，puts()函数会在字符串后自动加上换行符。

【例 6.12】

```
#include <stdio.h>
#define STRING1 "This is a string1"
int main(void)
{
    char str2[]="This is a string2";
    puts(STRING1);
    puts(str2);
    puts("This is a string3");
    return 0;
}
```

```
This is a string1
This is a string2
This is a string3
```

图 6-16　例 6.12 运行结果

运行结果如图 6-16 所示。

程序分析：三个 puts()函数分别输出符号常量、字符数组和字符串常量。显示效果如图 6-16 所示。

2．字符串输入函数 gets()

格式：gets(字符数组名)

功能：从标准输入设备（一般是键盘）输入一个字符串，使用回车表示字符串输入结束。本函数需要字符串地址作为参数用来保存输入字符串。

【例 6.13】

```
#include <stdio.h>
#define SIZE 20
int main(void)
{
    char name[SIZE];
    printf("input your name please:\n");
    gets(name);
    printf("%s , nice to meet you!\n",name);
    return 0;
}
```

```
input your name please:
Tom
Tom , nice to meet you!
```

图 6-17　例 6.13 运行结果

运行结果如图 6-17 所示。

程序分析：可以看出当输入的字符串中含有空格时，输出仍为全部字符串。说明 gets()函数并不以空格作为字符串输入结束的标志，而只以回车作为输入结束的标志，这与 scanf()函数是不同的。显示效果如图 6-17 所示。

3．字符串长度函数 strlen()

格式：strlen(字符串)

功能：返回字符串的长度(不含字符串结束标志'\0')。

【例 6.14】

```
#include <stdio.h>
#include <string.h>
int main(void)
{  int l;
   char str[]="To learn C language is happy!";
```

```
    l=strlen(str);
    printf("The length of the string is %d\n",l);
    return 0;
}
```

```
The lenth of the string is 29
```

图 6-18　例 6.14 运行结果

运行结果如图 6-18 所示。

程序分析：可以看出 strlen()函数只返回字符数组的有效字符个数，不包括字符串结束标志。显示效果如图 6-18 所示。

4．字符串连接函数 strcat()

格式：strcat (字符串 1, 字符串 2)

功能：将字符数组 2 的一个复制添加到第一个字符串后，并删去原字符串 1 后的串标志'\0'。本函数返回值是连接后的字符串 1 的地址。

【例 6.15】

```
#include <stdio.h>
#include <string.h>
int main(void)
{
    static char str1[30]="Your name is";
    char str2[10];
    printf("input your name:\n");
    gets(str2);
    strcat(str1,str2);
    puts(str1);
    return 0;
}
```

```
input your name:
Tom
Your name is Tom
```

图 6-19　例 6.15 运行结果

运行结果如图 6-19 所示。

程序分析：用来保存字符串 1 的字符数组应该有足够的长度用来保存连接后的字符串，否则会导致程序出错。显示效果如图 6-19 所示。

5．字符串复制函数 strcpy()

在字符串操作中，字符串复制是比较常用的操作之一，在字符串处理函数中包含 strcpy()函数，该函数可用于复制特定长度的字符串到另一个字符串中。

格式：strcpy (字符串 1, 字符串 2)

功能：将字符串 2 复制到字符串 1 中。字符串 2 可以是一个字符串常量，也可以是保存字符串的字符数组名。

【例 6.16】字符串复制。

```
#include<stdio.h>
#include<string.h>
int main()
{   char str1[30],str2[30];
    printf("输入目的字符串:\n");
    gets(str1);                          /*输入目的字符*/
    printf("输入源字符串:\n");
    gets(str2);                          /*输入源字符串*/
```

```
    printf("输出目的字符串:\n");
    puts(str1);                          /*输出目的字符*/
    printf("输出源字符串:\n");
    puts(str2);                          /*输出源字符串*/
    strcpy(str1,str2);                   /*调用 strcpy 函数实现字符串复制*/
    printf("调用 strcpy 函数进行字符串复制:\n");
    printf("复制字符串之后的目的字符串:\n");
    puts(str1);                          /*输出复制后的目的字符串*/
    return 0;                            /*程序结束*/
}
```

图 6-20　例 6.16 运行结果

运行结果如图 6-20 所示。

程序分析：在 main()函数体中定义两个字符数组，分别为存储源字符串和目的字符数组，然后获取用户为两个字符数组赋值的字符串，并分别输出两个字符数组，调用 strcpy()函数将源字符数组中的字符串赋值给目的字符数组，最后输出目的字符数组。

注意：

（1）要求目的字符数组有足够的长度，否则不能全部装入所复制的字符串。

（2）“目的字符数组”必须写成数组名形式；而“源字符数组名”可以是字符数组名，也可以是一个字符串常量，这时相当于把一个字符串赋予一个字符数组。

（3）不能用赋值语句将一个字符串常量或字符数组直接赋给一个字符数组，不能使用 str1=str2 语句将 str2 赋值给 str1，这是初学者易犯的错误，字符串的赋值需要使用 strcpy()函数实现字符串之间的复制，这是初学者易犯的错误。显示效果如图 6-20 所示。

6．字符串比较函数 strcmp()

格式：strcmp(字符串 1, 字符串 2)

功能：按照英文字典顺序比较字符串 1 和字符串 2，返回整型值表示比较结果。

返回值=0；字符串 1=字符串 2

返回值>0；字符串 1>字符串 2

返回值<0；字符串 1<字符串 2

【例 6.17】

```
#include <stdio.h>
#include <string.h>
int main(void)
 {
   char str1[] = "MyPassWord";
   char str2[11];
    printf("请输入密码:");
    gets(str2);
    if(!strcmp(str1, str2))
        printf("密码输入正确!\n");
    else
        printf("密码输入错误!\n");
    return 0;
}
```

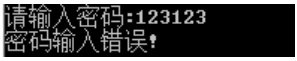


图 6-21 例 6.17 运行结果

运行结果如图 6-21 所示。

分析：定义两个字符数组 str1 用来存放正确密码，str2 用来存放新输入的密码，使用字符串比较函数来比较两字符串，若返回值为零则说明新输入的密码正确，否则密码错误。显示效果如图 6-21 所示。

6.4.7 字符数组程序设计举例

【例 6.18】判断所输入的用户名和密码。若用户名和密码匹配，则进入系统；若用户名或密码不正确，则提示错误并重新输入；若错误输入超过 3 次，则退出系统。

```
#include<stdio.h>
#include<string.h>
int main()
{  char user[20] = {"mrsoft"};
   char password[20] = {"mrkj"};
   char ustr[20],pwstr[20];
   int i=0;
   while(i < 3)
   { printf("输入用户名字符串:\n");
     gets(ustr);                          /*输入用户名字符串*/
     printf("输入密码字符串:\n");
     gets(pwstr);                         /*输入密码字符串*/
     if(strcmp(user,ustr))                /*如果用户名字符串不相等*/
     {printf("用户名字符串输入错误! \n"); /*提示用户名字符串输入错误*/}
     else                                 /*用户名字符串相等*/
     {if(strcmp(password,pwstr))          /*如果密码字符串不相等*/
        {printf("密码字符串输入错误! \n"); /*提示密码字符串输入错误*/}
        else                              /*用户名和密码字符串都正确*/
        {printf("欢迎使用! \n");          /*输出欢迎字符串*/
        break;
        }
     }
     i++;
   }
   if(i == 3)
   {printf("输入字符串错误 3 次! \n");    /*输入字符串错误 3 次*/}
   return 0;                              /*程序结束*/
}
```

运行结果如图 6-22 所示。

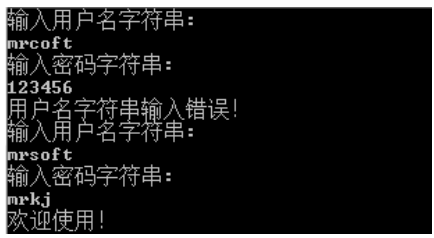


图 6-22 例 6.18 运行结果

程序分析：本程序使用 while 循环来完成用户名和密码的录入和验证，循环语句最多执行三次，若用户名密码匹配则输出欢迎语句，并且使用 break 语句跳出循环。若用户名或密码错误则进行下一次循环，重新输入用户名和密码，循环最多执行三次。最后判断循环次数是否超过三次，若超过三次提示错误次数超过限制次数，退出系统。显示效果如图 6-22 所示。

6.5　数组的应用程序设计举例

【例 6.19】反转输出字符串。

字符串操作在应用程序中经常会使用，如连接两个字符串、查找字符串等。本题需要实现的功能是反转字符串。以字符串“mrsoft”为例，其反转的结果为“ifosrm”。在本例的 main()函数体中定义两个字符数组，分别为源字符数组和目标字符数组，然后在循环遍历源字符数组的同时，将读取的字符从目标字符数组的末尾开始向前插入，最后分别输出源字符数组和目标字符数组。代码如下：

```
#include <stdio.h>
int main()
{
    int i;
    char String[7] = {"mrsoft"};
    char Reverse[7] = {0};
    int size;
    size = sizeof(String);              /*计算源字符串长度*/
    /*循环读取字符*/
    for(i=0;i<6;i++)
    {
        Reverse[size-i-2] = String[i]; /*向目标字符串中插入字符*/
    }
    /*输出源字符串*/
    printf("输出源字符串:%s\n",String);
    /*输出目标字符串*/
    printf("输出目标字符串:%s\n",Reverse);
    return 0;                           /*程序结束*/
}
```

运行结果如图 6-23 所示。

图 6-23　例 6.19 运行结果

分析：在程序中定义两个字符数组，一个表示源字符串，另一个表示反转后的字符串。在源字符串中，我们从第一个字符开始遍历，读取字符数据，在目标字符串中从最后一个字符（结束标记\0 除外）倒序遍历字符串，依次将源字符串中的第一个字符数据写入目标字符串的最后一个字符中，将源字符串中的第 2 个字符数据写入目标字符串的倒数第 2 个字符中。依此类推，这样就实现了字符串的反转。显示效果如图 6-23 所示。

【例 6.20】计算字符串中有多少个单词。

在本实例中输入一行字符，然后统计其中有多少个单词，要求每个单词之间用空格分隔开，且最后的字符不能为空格。

```
#include<stdio.h>
int main()
{
    char cString[100];                  /*定义保存字符串的数组*/
    int iIndex, iWord=1;                /*iWord 表示单词的个数*/
    char cBlank;                        /*表示空格*/
```

```
    gets(cString);                              /*输入字符串*/
    if(cString[0]=='\0')                        /*判断如果字符串为空的情况*/
    {
        printf("There is no char!\n");
    }
    else if(cString[0]==' ')                    /*判断第一个字符为空格的情况*/
    {
        printf("First char just is a blank!\n");
    }
    else
    {
        for(iIndex=0;cString[iIndex]!='\0';iIndex++) /*循环判断每一个字符*/
        {
            cBlank=cString[iIndex];             /*得到数组中的字符元素*/
            if(cBlank==' ')                     /*判断是不是空格*/
            {
                iWord++;                        /*如果是则加 1*/
            }
        }
        printf("%d\n",iWord);
    }
    return 0;
}
```

```
This is a cup!
4
```

图 6-24　例 6.20 运行结果

运行结果如图 6-24 所示。

分析：按照要求使用 gets()函数将输入的字符串保存在 cString 字符数组中。首先要对输入的字符进行判断，在数组中的第一个输入字符如果是结束符或空格，那么进行消息提示；如果不是则说明输入的字符串是正常的，这样就在 else 语句中进行处理。使用 for 循环判断每一个数组中的字符是否为结束符，如果是，则循环结束；如果不是，则在循环语句中判断是否为空格，遇到一个空格则对单词计数变量 iWord 进行自加操作。显示效果如图 6-24 所示。

本 章 小 结

数组是一组具有相同名字、下标不同的同类型变量的集合，主要用来处理问题中集合形式的数据，这是 C 语言中为什么需要引入数组的原因，也是数组需要理解的核心之一。

数组根据维数可分为一维数组、二维数组、多维数组等。不同维次的数组一般处理对应维次的集合数据，如一维数组处理队列、栈等一维形式的数据，二维数组处理表格、矩阵等二维形式的数据。

关于数组的主要知识点有数组的定义及初始化方式、数组元素的引用、字符数组与字符串、字符串处理函数。学习时以一维数组为基础，在掌握一维数组的基础上对比学习二维数组和字符数组。

字符串处理是程序设计语言中的重要组成部分，C 语言中是通过字符数组来处理字符串的，为了与普通字符数组区别，保存字符串的数组在最后加上结束标志'\0'，处理字符串的程序以此标志为界限来读取字符串。

为了便于处理字符串，C 语言提供了标准的字符串处理函数，要熟练使用这些函数，提高在程序中处理文本和字符串的能力。

本章介绍了数组的基础知识，在后续章节中，还要学习数组与指针、数组与函数的结合应用。

习　题　6

一、选择题

1．在 C 语言中，引用数组元素时，其数组下标的数据类型允许是（　　）。

A．整型常量　　B．整型常量或整型表达式

C．整型表达式　　D．任何类型的表达式

2．以下对一维整型数组 a 的正确说明是（　　）。

A．int a(10);　　B．int n=10,a[n];

C．int n; scanf("%d",&n); int a[n];　　D．#define SIZE 10　　int a[SIZE];

3．若要求定义具有 10 个 int 型元素的一维数组 a，则以下定义语句中错误的是（　　）。

A．#define N 10　　B．#define n 5

int a[N];　　int a[2*n];

C．int a[5+5];　　D．int n=10,a[n];

4．若有定义：int a[10]，则对数组 a 元素的正确引用是（　　）。

A．a[10]　　B．a[3.5]　　C．a（5）　　D．a[10-10]

5．以下能对一维数组 a 进行正确初始化的语句是（　　）。

A．int a[10]=(0,0,0,0,0);　　B．int a[10]={　};

C．int a[ ]={0};　　D．int a[5]={0,1,2,3,4,5};

6．不是给数组的第一个元素赋值的语句是（　　）。

A．int a[2]={1};　　B．int a[2]={1*2};

C．int a[2];scanf ("%d",a);　　D．a[1]=1;

7．下列定义正确的是（　　）。

A．static int a[6]={1,2,3,4,5};　　B．int b[ ]={2.5};

C．int a(10);　　D．int 4e[4];

8．以下不正确的定义语句是（　　）。

A．double x[5]={2.0,4.0,6.0,8.0,10.0}；　　B．int y[5]={0,1,3,5,7,9} ;

C．char c1[ ]={'1','2','3','4','5'} ;　　D．char c2[ ]={'\x10','\xa','\x8'} ;

9．假设 array 是一个有 10 个元素的整型数组，则下列写法中正确的是（　　）。

A．array[0]=10;　　B．array=0;　　C．array[10]=0;　　D．array[-1]=0;

10．执行以下程序段后，a 的值是（　　）。

```
static int a[]={5,3,7,2,1,5,4,10};
    int a=0,k;
    for(k=0;k<8;k+=2)
    a+=a+k;
```

A．17　　B．27　　C．13　　D．有语法错误，无法确定

二、判断题

1．在 C 语言中，数组元素的下标是整型常量或整型变量，并且从 1 开始。　　（　　）

2．数组元素的值可以使用赋值语句或输入函数进行赋值，但占用运行时间。 （ ）

3．对一维数组初始化时，数组的长度可以省略，系统会自动按初值的个数分配存储空间。（ ）

4．在初始化数组时，若指明了数组的长度，而提供的常量个数小于数组的长度，则只给相应的数组元素赋值，其余无值。 （ ）

5．在初始化数组时，若数组长度小于初值的个数，则会产生编译错误。 （ ）

6．如果对数组不赋初值，则数组元素一定取随机值。 （ ）

7．二维数组在内存中存储是以列为主序方式存放，即在内存中先存放第一列的元素，再存放第二列的元素。 （ ）

8．定义二维数组时，若对全部元素都赋初值，则第一维的长度不能省，但第二维的长度可以不指定。 （ ）

9．数组名的命名规则与变量名不相同。 （ ）

10．在声明一数组时，需要给出所包含数组元素的个数，即数组长度，这个长度是可改变的。（ ）

11．在声明一数组时，可用一个整型变量来表示数组的长度。 （ ）

12．字符个数多的字符串比字符个数少的字符串大。 （ ）

13．在 C 语言中，二维数组可给出所有元素赋初值，也可以为部分元素赋初值。 （ ）

14．有数组定义 int a[2][2]={{1},{2,3}};则 a[0][1]的值为 0。 （ ）

15．若有以下的数组定义：

char x[]="12", y[]={'1','2'}; 则 x 数组和 y 数组长度相同。 （ ）

16．if(strl>str2)printf("%s",strl);else printf("%s",str2);表示输出较大字符串。 （ ）

17．判断两个串的大小，可以根据字符串的长度来判断。 （ ）

三、编程题

1．编程实现如下功能：输入 N×N 的阶矩阵，然后求出其主、次对角线元素之和。

2．定义一个包含 10 个元素的整型数组，输入 10 个整数并求数组元素的和。

3．在屏幕上打印下列矩阵。

```
1 2 2 2 2 2 1
3 1 2 2 2 1 4
3 3 1 2 1 4 4
3 3 3 1 4 4 4
3 3 1 5 1 4 4
3 1 5 5 5 1 4
1 5 5 5 5 5 1
```

4．编程实现将一段英文文本的小写字母全部转换成大写字母。

5．有三个字符串，找出其中最大者。

第7章　函　　数

7.1　函 数 概 述

一个 C 程序一般都是由一个主函数和若干个辅助函数构成的。在 C 语言中，函数是程序的最基本的执行单位。一个 C 程序若没有函数就无法执行，也就不能称其为 C 程序了。作为用语言编写的一个项目，通常也不把所有的函数都放在一个源程序文件中，而是依据所实现的功能将其放在多个文件中，可以由多人共同完成一个项目。这样的分工合作更便于一个项目的完成。

所以说，一个 C 语言源程序可由多个源程序文件构成；而每个文件可由多个完成不同功能的函数构成；每个函数则由若干行程序语句组成。在这里，函数是程序的基本构成单位，语句是程序的最小构成单位。

需要注意的是，一个 C 程序有且只有一个主函数 main()，程序执行都是从主函数开始的。也就是说，只有找到这个主函数，程序才能正常执行。包含主函数在内的所有函数之间的关系是平等的，即 C 语言中的函数可以按任何顺序出现在 C 程序中，函数之间没有包含与被包含的关系，只有主函数与辅助函数及辅助函数之间的调用与被调用的关系。但是，一个函数定义不允许重复出现在一个 C 程序中。

从用户使用角度来看，函数分为以下两类。

（1）标准函数（也称库函数或系统函数）：由系统提供，是一些常用功能模块的集合。每个标准函数都已经被设计好完成某种功能，如 printf()和 scanf()分别完成数据的输入和输出功能。用户若想用这些功能，就不必再编写代码了，只需将这个函数所在的头文件用#include 宏命令包含进程序中就可以了。值得注意的是，每个 C 版本所提供的系统函数的功能和数量都不尽相同，使用时要查看相应的函数说明。

（2）用户自定义函数：系统函数是不可能把所有功能都考虑进去的，一个项目中的绝大多数功能都需要用户自己编写代码。这也是 C 语言中体现用户编程能力的最重要的一环，也正是我们编程人员必须掌握的基本能力。

从函数的形式来看，函数分为以下两类。

（1）无参函数。例 7.1 中的 printmessage 和 printstar 就是无参函数。在调用无参函数时，调用函数并不将数据传递给被调用函数。

（2）有参函数。例 7.2 中的 area 就是有参函数。在调用有参函数时，调用函数和被调用函数之间有数据传递。也就是说，调用函数将数据传递给被调用函数使用，被调用函数中的数据也可以带回到调用函数使用。

下面通过两个程序来简单地介绍函数的作用。

【例 7.1】简单函数调用示例。

```
main()
{  printstar();
   printmessage();
   printstar();
```

```
}
printstar()
{ printf("\t***********\n");
}
printmessage()
{ printf("\t* Welcome *\n");
}
```

运行结果如下：

```
***********
* Welcom  *
***********
```

分析：此例中的 printstar()和 printmessage()两个函数都是无参函数，无返回值。将两个函数的位置放在主函数之前，运行结果也是一样的。

【例 7.2】通过输入半径值，计算圆的面积。

```
float area(float x)
{  float s;
   s=3.14159*x*x;
   return (s);
}
main()
{  float r,s;
   printf("Enter r:"); scanf("%f",&r);
   s=area(r);
   printf("s=%f\n",s);
}
```

运行结果如下：

```
Enter r:10↙
s=314.158997
```

分析：本例中的 area()函数是有参函数，放在了主函数之前。是否可将它放在主函数之后?这个问题将在 7.4 节中介绍。

7.2 函 数 定 义

函数的定义就是要确定一个函数完成什么样的功能以及怎样运行。函数定义的一般形式如下：

```
函数存储类别 函数返回值类型 函数名(函数形式参数表)
{ 函数体说明部分
  函数功能语句序列
}
```

说明：

（1）函数存储类别：表明该函数是外部函数，还是内部函数。

若是 extern 标识符，则表明该函数是外部函数，可以被程序中的其他函数调用。若是 static 标识符，则表明该函数是内部函数，只能在定义该函数的文件中被调用。若省略函数的存储类别，则系统默认的存储类别为 extern。

（2）函数返回值类型：表明调用该函数时将带回一个何种类型的值。

return 语句中的表达式类型一定要与该类型相一致。若省略函数的返回值类型，则系统默认为 int。值得注意的是，void 类型的函数没有返回值。

（3）函数名：是代表该函数的一个标识符。

使用函数的时候，一定要通过函数名称标识。该名称要符合标识符的起名规则，虽然可随意起名，但也要尽量做到见名知义。

（4）函数形式参数表：这是函数内部与其他函数联系的一种桥梁和纽带。函数外部信息通过形式参数表传入函数体内，完成函数指定的功能。

函数形式参数表的格式为：参数 1，参数 2，参数 3，…

函数形式参数说明的格式为：类型 1　参数 1，类型 2　参数 2，…

两者可以合起来统一放在形式参数表中，这时，每个参数都必须独立说明其类型。例如：

```
int add(int x,int y)
{ int c;
c=x+y;
return(c);
}
```

与下面函数定义是等价的：

```
int add(x,y)
int x,y;
{ int c;
c=x+y;
return (c);
}
```

（5）函数体说明部分：定义说明一些实现函数功能的变量和类型等。如上面的变量 c 的定义。

（6）函数功能语句序列：实现函数功能的语句的有机集合。

这是函数的主体，只有按功能编写相应的语句行，才能最终实现整个程序的功能。

7.3　函 数 调 用

7.3.1　函数调用的一般形式

函数只有被调用才能真正地执行。调用函数就是通过函数名把指定的参数传送到函数中去，使得该函数带着指定的参数值完成具体的函数功能。函数调用的一般形式为：

函数名(实际参数表)

如果被调用函数是无参函数，则没有“实际参数表”，但括号不能省略。如果“实际参数表”包括两个或两个以上实际参数，则参数之间用逗号隔开。另外，参数的类型名不能写上，参数的类型和个数一定要与形式参数表中的各个参数被说明的类型和参数个数一一对应、保持一致。

7.3.2　函数调用的方式

按函数在程序中出现的位置，函数调用有以下三种方式。

1．函数语句

把函数调用作为一个语句。这时不要求函数带回值，只要求函数完成一定的操作。例如，例 7.1 中的语句：

```
printmessage();
```

2．函数表达式

函数出现在一个表达式中，这时要求函数带回一个确定的值以参加表达式的运算。例如：

```
m=add(a,b)/2
```

其中，函数 add(a,b)是表达式的一部分，它的值除以 2 再赋给 m。

3．函数参数

函数调用作为一个函数的参数，其实质就是将函数的返回值作为这个函数的一个实参。例如：

```
m=add(add(a,b),c);
```

其中，add(a,b)是一次函数调用，它的值作为函数 add 另一次调用的实参，m 的值是 a、b、c 三者之和。又如：

```
printf("%d\n",add(a,b));
```

把 add(a,b)作为 printf()函数的一个实参。

【例 7.3】通过调用函数计算任意三个整数的和。

```
int add(x,y,z)
int x,y,z;
{ return x+y+z;
}
main()
{  int a,b,c;
   printf("Input a,b,c:");
   scanf("%d%d%d",&a,&b,&c);
   printf("add=%d\n",add(a,b,c));
}
```

运行结果如下：

```
Input a,b&c:4 5 6↙
add=15
```

7.4 函数引用说明

一个函数在被另一个函数调用时，首先被调用函数必须存在。另外，如果被调用函数是系统函数，则应在源程序文件开头用#include 命令将被调用的库函数所在头文件（扩展名为.h）包含到本文件中。如果被调用函数是用户自定义函数，则应在调用函数的说明部分加上引用说明。所谓引用说明，就是对函数的名称、返回值类型以及形参的类型和顺序加以说明。引用说明的主要作用是，利用它在程序的编译阶段对被调用函数的合法性进行全面检查，如函数名是否正确，函数返回值的类型，形式参数与实际参数的类型和个数是否一致等。函数引用说明的形式为：

函数返回值类型 函数名(类型 1 形参 1,类型 2 形参 2,…);

其中，形式参数表也可以省略，写成：

函数返回值类型 函数名(类型 1,类型 2,…);

当参数类型为 int 或 char 时，甚至类型表也可以省略，而写成：

函数返回值类型 函数名();

这是最终的省略表示方法，不能再省略了，尤其是不能省略最后的一对圆括号和分号。

例如，例 7.3 中的主函数 main()应该修改为：

```
main()
{ int a,b,c;
   int add();                              /*函数引用说明*/
   printf("Input a,b,c:");
   scanf("%d%d%d",&a,&b,&c);
   printf("add=%d\n",add(a,b,c));          /*函数调用*/
}
```

那么，为什么这个程序不用加引用说明也能正确执行呢？其实，在 Turbo C 中，有下列几种情况可以省略函数引用说明。

（1）函数的返回值类型为 int 或 char。省略的类型为 int。

（2）函数的定义写在主调函数之前。

（3）在所有函数定义的前面已加了函数引用说明。

例 7.3 既符合（1），也符合（2），所以就可以省略函数引用说明了。

另外，请读者注意函数定义与函数引用说明的区别。

7.5　函数的参数和返回值

7.5.1　形式参数和实际参数

定义函数时的参数称为形式参数，简称形参。调用函数时的参数称为实际参数，简称实参。在调用有参函数时，首先进行参数传递。参数传递的规则是，实参表达式的值依次对应地传给形参表中的各个形参变量。这种实参到形参的传递是单向的值传递。确切地说，在函数被调用时，系统为每个形参分配存储单元，然后把相应的实参值传送到这些存储单元作为形参的初值，再执行规定的操作。这种传值操作的特点是：函数中对形参的操作，不会影响到调用函数中的实参变量，即形参的值不能传回给实参。

【例 7.4】函数调用参数传递举例。

```
swap(int x,int y)                    /*函数定义*/
{  int t;
   t=x;x=y;y=t;
   printf("x=%d, y=%d\n",x,y);
}
main()
{  int a=10,b=20;
   printf("a=%d, b=%d\n",a,b);
   swap(a,b);                        /*函数调用*/
```

```
    printf("a=%d, b=%d\n",a,b);
}
```

运行结果如下：

```
a=10,b=20
x=20,y=10
a=10,b=20
```

说明：

（1）在函数调用之前，形参并不占内存中的存储单元，只有在发生函数调用时，形参才被分配内存单元，函数调用结束后，形参所占的内存单元也被释放。

（2）形参是变量，实参是表达式。例如，要调用函数 add()，下面几种形式都是正确的。

```
add(10,5);   add(a,b);    add(a+10,b);    add(10,a*b);
```

（3）形参与实参的类型要一致，否则按不同类型数值的赋值规则进行转换。

（4）C 语言规定，实参对形参的数据传递是单向值传递，即只由实参传给形参，而不能由形参传回来给实参。在内存中，实参变量与形参变量分占不同的存储单元。

7.5.2 函数的返回值

在大多情况下，希望通过函数调用返回一个有确定意义的值，这个值就是函数的返回值。C 语言中，通过 return 语句将被调用函数中的一个确定值带回到调用函数中。return 语句的格式为:

```
return 表达式;
```

或

```
return (表达式);
```

return 语句的功能：从函数中退出，返回到调用它的函数中，同时带回表达式的值。

说明：

（1）一次函数调用最多只能得到一个返回值。一个函数中可以有多个 return 语句，但只有第一个被执行到的 return 语句起作用。

（2）函数返回值的类型是定义函数时指定的类型。若 return 语句中表达式值的类型与定义函数时指定的类型不一致，则以定义函数时的类型为准。对数值型数据，自动进行类型转换。

（3）如果被调用函数中没有 return 语句，函数并不是不带回值，只是带回一个不确定的无用值。例如，在例 7.4 中，尽管没有要求函数 swap 带回值，但如果主函数 main()修改为：

```
main()
{  int a=10,b=20,c;
   printf("a=%d, b=%d\n",a,b);
   c=swap(a,b);
   printf("a=%d, b=%d\n",a,b);
   printf("c=%d \n",c);
}
```

运行时除了得到和例 7.4 一样的结果外，还输出：c=11（11 是一个随机值）。

（4）若明确表示函数调用不带回值，则在定义函数时，类型指定为 void。这样，系统就保证不使函数带回任何值，即不允许在函数中用 return 语句来返回值。在例 7.4 中，如果将函数 swap 的类型定义为 void，则下面的用法是错误的：

```
c=swap(a,b);
```

编译时会给出出错信息。

7.5.3　指针作为函数参数

1．普通变量的指针与函数参数

将普通变量的地址传递给形参变量，形参变量必须是指针类型的。指针作为函数参数进行传递，实质上还是值的单向传递，传过来的值只不过是个地址，实参和形参这两个量将指向同一个存储单元。在函数中对形参变量所指向内存单元的值的改变就相当于改变实参所指向的内存单元的值。严格地说，并不是改变了实参和形参指针变量的值，而是将两个指针变量所指向的同一个地址单元中的值改变了。在程序设计过程中，程序员往往利用这点，在编写程序时，通过函数调用带回多个值。

【例 7.5】编写函数实现将 a 和 b 的值交换。

```
void swapab(int *x,int *y)
{  int z;
   z=*x; *x=*y; *y=z;
}
main()
{  int a=10,b=20;
   printf("Before swap:a=%d,b=%d\n",a,b);
   swapab(&a,&b);
   printf("After swap: a=%d,b=%d\n",a,b);
}
```

运行结果如下：

```
Before swap:a=10,b=20
After swap: a=20,b=10
```

思考题：若将函数 swapab()写成如下形式，能完成以上功能吗？

```
void swapab(int *x,int *y)
{ int *z;
  z=x; x=y; y=z;
}
```

若改成如下形式呢？

```
void swapab(int *x,int *y)
{  int *z;
   *z=*x; *x=*y; *y=*z;
}
```

2．数组地址与函数参数

前面讨论了数组，数组是存储数据的重要工具。因为数组中存放的数据有先后的次序关系，故很容易进行统一处理。而函数是构成程序的基本单位，它是可以通过参数传递来处理数组的。在参数传递中，可以把实参数组的地址直接传递给形参指针变量，然后在函数中处理数组元素。形参指针变量将指向实参数组的起始地址或分量的地址。

1）一维数组作为函数参数

【例 7.6】将一维数组中的第一个元素与最后一个元素交换位置。

```
exchangefl(int *x,int n)
{  int t;
   t=*x; *x=x[n-1]; x[n-1]=t;
}
main()
{  int a[8],i;
   for(i=0;i<8;i++) scanf("%d",a+i);
   exchangefl(a,8);
   for(i=0;i<8;i++) printf("%d\t",a[i]);
   printf("\n");
}
```

运行结果如下：

```
1 2 3 4 5 6 7 8↙
8   2   3   4   5   6   7   1
```

将函数 exchangefl()说明部分改成如下形式也是一样的：

```
exchangefl(int x[],int n)
```

在这里，x 不是数组名，而是指针变量名。它写成数组形式也是允许的。无论写成什么形式，在函数内处理数组分量时，使用指针形式(*x)也行，使用数组分量形式(x[n-1])也对。希望读者们要注意体会这一点。

2）二维数组作为函数参数

【例 7.7】将 3×5 数组中的最大值与最小值互换位置。

```
exchangemm(int x[][5])
{ int i,j,max,min,hi,hj,li,lj,t;
  max=min=x[0][0];hi=hj=li=lj=0;
  for(i=0;i<3;i++)
    for(j=0;j<5;j++)
    { if(x[i][j]>max) { max=x[i][j];hi=i;hj=j;}
      if(x[i][j]<min) { min=x[i][j];li=i;lj=j;}
    }
  t=x[hi][hj]; x[hi][hj]=x[li][lj]; x[li][lj]=t;
}
main()
{ int a[3][5],i,j;
  for(i=0;i<3;i++)
    for(j=0;j<5;j++)
      scanf("%d",&a[i][j]);
  exchangemm(a);
  for(i=0;i<3;i++)
  { for(j=0;j<5;j++)
      printf("%d\t",a[i][j]);
    printf("\n");
  }
}
```

运行结果如下：

```
22 18 29 17 45↙
30 89 72 56 61↙
87 93 67 25 36↙
22  18  29  93  45
30  89  72  56  61
87  17  67  25  36
```

上面的函数 enchangemm()中的形参写成 int x[][5]，也可能写成 int x[3][5]和 int (*x)[5]。若写成 int *x[3]和 int **x，对程序要做一些调整，请读者自己做实验完成。如果函数里用指针来完成，可能会简单一些。例如，函数 exchangemm()改成：

```
exchangemm(int x[][5])
{ int *p,*q,*r,t;
  p=q=x[0];
  for(r=x[0]+1;r<x[0]+3*5;r++)
  { if(*r>*p) p=r;
    if(*r<*q) q=r;
  }
  t=*p;*p=*q;*q=t;
}
```

在此，值得注意的是：x 是二级指针，而 p,q,r 及 x[0]都是一级指针。另外，多维数组在内存中是按行优先顺序存储的。

3）字符串作函数参数

字符串是用于存储处理非数字信息的数据。实际上，字符串就是数组的形式，只是它在处理问题时有其自身的独特之处，所以也在这里单独讨论一下。

【例 7.8】将一字符串倒过来与原字符串连接成一个字符串。

```
strcatrep(char *s)
{ char *p,*q;
  p=s;
  while(*p!='\0') p++;
  q=p-1;
  while(q>=s) *p++=*q--;
  *p=\0;
}
main()
{ char str[80];
  gets(str);
  strcatrep(str);
  puts(str);
}
```

运行结果如下：

```
abcdefghij↙
abcdefghijjihgfedcba
```

值得注意的是，每个字符串都有一个结束标志：'\0'，在编程时要充分利用这一点。

7.5.4 主函数与命令行参数

无论是多么复杂的 C 程序，总有且仅有一个主函数 main()，它担负着程序执行起点的作用，是可执行的 C 程序不可缺少的函数。

主函数的格式为：

```
main(int argc,char *argv[])
{…}
```

从形式上看，除对括号里的内容没介绍外，对其他的内容，读者都已经熟悉了。下面对括号内的信息加以介绍。

我们称括号里的信息为命令行参数。对一个已经通过编译连接生成可执行文件的 C 程序来说，当用户在 DOS 方式下执行该程序时，有时要输入一些参数。命令行参数用于接收这些参数，把它们带到主函数中去完成程序的执行。

其中，argc 用于保存用户命令行中输入的命令中参数的个数，命令名本身也作为一个参数。argv[]是一个字符指针数组，它用于保存各个参数的名字(包括命令名本身)，每个参数名都是一个字符串。对命令名，系统将会自动加上盘符、路径、文件名，而且变成大写字母串存储到 argv[0]中。其他命令行参数名将会自动依次存入到 argv[1], argv[2], …, argv[argc-1]中。

【例 7.9】命令行参数简单示例。

```
main(int argc,char *argv[])
{ while(--argc>=0)
   puts(argv[argc]);
}
```

假设该程序文件名为：exam1.c，经过编译连接，最后生成了一个名为 EXAM1.EXE 的可执行文件。如果在命令状态下，我们输入命令行为：EXAM1 horse house monkey donkey friends，则该程序的输出将会是：

```
friends
donkey
monkey
house
horse
C:\TC\EXAM1.EXE
```

即倒序输出了命令行中的各个参数名。

思考题：要正序输出命令行中的各个参数，该怎么做？下面程序将会输出什么结果？

```
main(int argc,char *argv[])
{ while(argc!=0)
   puts(argv[--argc]);
}
```

通常，用 argc 和 argv 作为命令行参数中的形参名，用户也可以用其他的标识符作为形参变量名。

7.6 函数与带参数的宏的区别

在前面的章节中已介绍过带参数的宏，其一般形式为：

#define 宏名(参数表) 字符序列

其中，字符序列中一般要含有参数表中的参数。那么，它与函数的调用形式是否一样呢？事实上，它们有很大的区别，可以说它们不是一回事。

带参数的宏，是在系统编译之前就将实际参数替换了形式参数，然后原样展开到程序中，用指定参数的“字符序列”替换“宏名(参数表)”，之后程序才进行编译连接生成目标程序。而函数是先进行编译连接生成目标程序，然后，在函数调用时将实际参数的值传递给形式参数变量，参与函数的执行。下面通过一个程序对比一下。

【例 7.10】已知半径值，计算圆面积。

1．用函数完成

```
#define PI 3.1415926
float area(float x)
{ float y;  y=PI*x*x;  return y;}
main()
{  float r,s;
   printf("Input radius:"); scanf("%f",&r);
   s=area(r);
   printf("r=%f\ts=%f\n",r,s);
}
```

运行结果如下：

```
Input radius:10↙
r=10.000000     s=314.159271
```

2．用带参数的宏完成

```
#define PI 3.1415926
#define AREA(X)  PI*X*X
main()
{  float r,s;
   printf("Input radius:"); scanf("%f",&r);
   s=AREA(r);
   printf("r=%f\ts=%f\n",r,s);
}
```

运行结果如下：

```
Input radius:10↙
r=10.000000     s=314.159271
```

可以看出，两个程序的运行结果是完全一样的，好像带参数的宏还简单些。有时是这样的，但多数情况下带参数的宏会使程序看上去更复杂。用函数完成时，参数传递是在函数调用时进行的。也就是说，函数已经经过编译连接生成了目标程序，在调用函数时只是将实参表达式的值传递给形参变量，然后执行函数功能部分。在用带参数的宏来完成时，要在程序编译之前完成如下的宏展开（两个程序中定义的宏名常量 PI 都要先展开）：

```
s=AREA(r);          s=PI*r*r;           s=3.1415926*r*r;
```

然后，再对展开后的程序进行编译连接生成目标程序。所以，从内部处理的过程来看，它们

绝对是不同的。另外，在定义带参数的宏时，宏参数通常要用括号括上，不然很容易出错。比如，上面的宏调用：s=AREA(r); 若改成：s=AREA(r+2);则程序的运行结果中，r=10.000000，s=53.415928。为什么半径大了，反而面积小了呢？原因是宏展开后变成了如下语句：

```
s=AREA(r+2);
s=PI*r+2*r+2;
s=3.1415926*r+2*r+2;
```

也就是说，在宏展开时，宏实参串原样传递给宏形参，宏形参再原样展开到后面的字符序列中去，不会自动地把宏形参在字符序列中加上括号的。所以上面的带参数的宏定义应该写成：

```
#define AREA(r) PI*(r)*(r)
```

这样展开后的结果就变成：

```
s=3.1415926*(r+2)*(r+2)
```

当然就不会出现上面的错误了。

总之，函数与带参数的宏比较，有以下几个不同点。

（1）函数调用时，先求出实参表达式的值，然后代入形参变量。而使用带参数的宏只是进行简单的字符替换。

（2）函数调用是在程序运行时处理的，分配临时的内存单元。而宏展开是在编译之前进行的，不分配内存单元，不进行值的传递，也没有返回值的概念。

（3）函数中的实参表达式和形参变量都有类型。而宏不存在类型之说，宏名无类型，宏参数也无类型。

（4）调用函数只能得到一个返回值，而宏调用可以设法得到多个结果。

【例 7.11】计算圆的周长，圆的面积和球的体积。

```
#define PI 3.1415926
#define CIRCLE(R,L,S,V)
L=2*PI*(R);
S=PI*(R)*(R);
V=4.0/3*PI*(R)*(R)*(R)
main()
{  float r,l,s,v;
   printf("Input radius:");  scanf("%f",&r);
   CIRCLE(r,l,s,v);
   printf("r=%6.2f, l=%6.2f, s=%6.2f, v=%6.2f\n",r,l,s,v);
}
```

运行结果如下：

```
Input radius:3.5↙
r=3.50, l=21.99, s=38.48, v=179.59
```

实际上，这个宏展开是展开了多个 C 语句，所以得到了多个计算结果。

（5）使用宏次数多时，宏展开后源程序长。因为每展开一次都使程序增长，而函数调用不会使源程序变长。

（6）宏替换不占用运行时间，只占用编译时间。而函数调用则占运行时间，不占编译时间。

使用带参数的宏一般会使程序看上去简短些，给程序设计带来一些方便，但容易出错。函数

通常完成一个预定功能，程序结构更紧凑些。另外，函数能实现所有带参数的宏所完成的功能，而带参数的宏则不能完成所有函数的功能。所以究竟用哪种形式要根据实际需要而定。

7.7　函数的嵌套调用与递归调用

7.7.1　函数的嵌套调用

函数是不允许嵌套定义的，但允许嵌套调用。所谓嵌套调用就是函数在被调用过程中又去调用了其他函数。嵌套调用其他函数的个数又称为嵌套的深度或层数。无论嵌套调用多少层，每个函数调用结束后都会返回到调用点，再继续程序的执行，直到主函数执行完成或遇到系统函数 exit() 调用强行结束程序执行。图 7-1 是函数嵌套调用示意图。

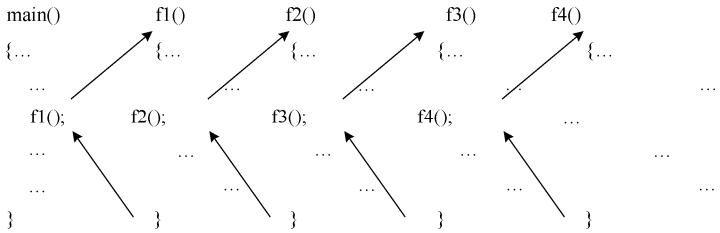


图 7-1　函数嵌套调用的示意图

原则上，对嵌套的层数没有限定。但在实际编程时，嵌套的层数不要过多，否则会使程序结构显得乱套，不易理解。

【例 7.12】函数嵌套举例。

```
void printmessage(void),printstart(void);
main()
{ printmessage();
}
void printmessage(void)
{ printstart();
  printf("\t* Welcome *\n");
  printstart();
}
void printstart(void)
{ printf("\t***********\n");
}
```

运行结果如下：

```
***********
* Welcome *
***********
```

7.7.2　函数的递归调用

函数的递归调用就是在调用一个函数的过程中直接或间接地调用了该函数本身。根据调用方式，递归调用又分为直接递归和间接递归两种。函数的递归调用实质上可以看成函数的嵌套调用的一种特例，如图 7-1 所示，若函数 f2(),f3(),f4()都是函数 f1()，就是直接递归。

递归不允许无限地执行下去，一定要有递归结束条件。当遇到递归结束条件时，递归程序将逐层往回返，直到递归调用结束。

【例 7.13】用递归法计算 n!。

计算公式为：$n!=\begin{cases}1 & n=0,1\\ n(n-1)! & n>1\end{cases}$

n=0 或 n=1 为递归结束条件。可以通过下推和回代两个过程描述（以求 5!为例），如图 7-2 所示。

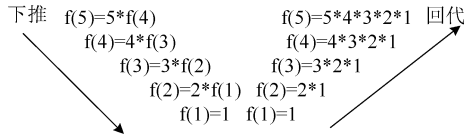


图 7-2 递归法应用举例

```
long f(int n)
{  long s;
   if(n<0) { printf("Error!"); exit(0);}
   if(n==0||n==1) s=1;
   else s=n*f(n-1);
   return s;
}
main()
{  int n; long m;
   printf("n=");scanf("%d",&n);
   m=f(n);  printf("%d!=%ld\n",n,m);
}
```

运行结果如下：

```
n=5↙
5!=120
```

递归调用在实际中采用很多，充分利用递归函数可以使程序结构简单、清晰，但增加了程序内部调用的复杂性。因为函数调用是通过系统内部的称为栈的结构来实现的，调用函数时将现场信息压栈，函数调用结束返回时将会把栈里保存的现场信息弹出，以使程序能知道从哪里开始，带着什么数据去继续执行。递归调用压栈和弹栈较频繁，这样就会增加系统的时间和空间复杂程度。所以，对于递归编程要根据需求，酌情考虑。

7.8 函数指针与返回指针的函数

7.8.1 函数指针

前面已介绍过，变量有地址，数组有地址，字符串有地址。那么，函数有没有地址呢？回答是肯定的，每个函数都占有相应的内存单元。它的起始地址由函数名来识别，即函数名是函数的入口地址，函数名是一个地址常量，也就是函数的指针。我们可以定义一个指向函数的指针变量，这个指针变量保存哪个函数的名称，它就指向哪个函数的地址，也就可以通过这个变量来调用函数。指向函数的指针变量的定义形式为：

```
函数返回值类型 (*指针变量名)();
```

【例 7.14】求两个整数的最大值、最小值、和、差、积等。

```
int max(int x,int y)
{ return x>y?x:y;}
int min(int x,int y)
{ return x<y?x:y;}
int add(int x,int y)
{ return x+y;}
int sub(int x,int y)
{ return x-y;}
int mult(int x,int y)
{ return x*y;}
main()
{ int a,b,(*fun)();
  scanf("%d%d",&a,&b);
  fun=max;
  printf("max=%d\n",(*fun)(a,b));
  fun=min;
  printf("min=%d\n",(*fun)(a,b));
  fun=add;
  printf("add=%d\n",(*fun)(a,b));
  fun=sub;
  printf("sub=%d\n",(*fun)(a,b));
  fun=mult;
  printf("mult=%d\n",(*fun)(a,b));
}
```

运行结果如下：

```
5 10↙
max=10
min=5
add=15
sub=-5
mult=50
```

7.8.2　函数指针作函数的参数

函数的指针也可以作为函数参数使用。我们将上面的例 7.14 增加一个函数 fun()，程序改写如下。

【例 7.15】求两个整数的最大值、最小值、和、差、积等。

```
int max(int x,int y)
{ return x>y?x:y;}
int min(int x,int y)
{ return x<y?x:y;}
int add(int x,int y)
{ return x+y;}
int sub(int x,int y)
{ return x-y;}
int mult(int x,int y)
```

```
{ return x*y;}
int fun(int (*f)(),int x,int y)
{ printf("%d\n",(*f)(x,y));
}
main()
{ int a,b;
  scanf("%d%d",&a,&b);
  fun(max,a,b);
  fun(min,a,b);
  fun(add,a,b);
  fun(sub,a,b);
  fun(mult,a,b);
}
```

运行结果如下：

```
5 10↙
10
5
15
-5
50
```

7.8.3 返回指针的函数

一个函数可以返回 int、char、float、double、void 等类型值，也可以返回指针类型值。这时的函数定义形式为：

```
函数返回值类型 *函数名(形式参数名表)
{ 函数体语句序列 }
```

【例 7.16】查找数组中的最大值。

```
int *max(int *x,int n)
{ int *p,*q;
  for(q=x,p=x+1;p<x+n;p++)
    if(*p>*q) q=p;
  return q;
}
main()
{ int a[10],i,*m;
  printf("Input 10 datas:");
  for(i=0;i<10;i++)
    scanf("%d",a+i);
  m=max(a,10);
  printf("max=%d\n",*m);
}
```

运行结果如下：

```
Input 10 datas:90 34 27 889 37 379 674 27 83 63↙
max=889
```

返回指针类型的函数通常在内存中开辟存储单元，然后把这个已开辟好的存储单元的地址返回。这一点在学“数据结构”课程时将会遇到很多，这里就不多叙了。

7.9　变量的作用域

变量的作用域是指变量在程序中起作用的范围。有的变量是在整个程序中起作用，有的变量是在一个文件中起作用，有的变量是在一个函数中起作用，而有的变量只是在一个小程序段中起作用。从作用域的角度，变量分为局部变量和全局变量。

7.9.1　局部变量

在函数内部定义的变量称为局部变量，也称内部变量。局部变量只在定义它的函数内有效，即只有定义它们的函数才能使用，不能被其他函数使用。

【例 7.17】局部变量举例。

```
main()
{ int a=2,b=3;          /*定义局部变量 a、b，只能在主函数 main 中使用*/
  int fun(int x);       /*函数引用说明 */
  printf("a=%d\n",a);   /*输出：a=2 */
  fun(b);               /*函数调用 */
  { int c=4;            /*定义局部变量 c，只能在该复合语句中使用*/
    c*=b;
    printf("c=%d\n",c); /*输出：c=12*/
  }
  printf("a=%d\n",a);   /*输出：a=2*/
}
int fun(int x)
{ int a=1,d=5;          /*定义局部变量 a、d、x，只能在函数 fun 中使用*/
  a+=x+d;
  printf("a=%d\n",a);   /*输出：a=9*/
}
```

运行结果如下：

```
a=2
a=9
c=12
a=2
```

说明：

（1）在主函数中定义的变量也是局部变量，只在主函数中有效，并不因为它在主函数中定义而在整个程序或文件中有效。另外，主函数也不能使用其他函数中定义的变量。

（2）不同函数中定义的局部变量可以同名，它们代表不同的对象，互不干扰。例如，在例 7.17 的主函数中定义了变量 a，在函数 fun()中也定义了变量 a，它们在内存中占有不同的单元，互不影响。

（3）形式参数也是局部变量，只在它所在的函数中有效，其他函数不能使用。例如，在例 7.17 中，函数 fun()的形参 x 是局部变量，只在函数 fun()中有效。

（4）在一个函数的内部，可以在复合语句中定义变量。在复合语句中定义的变量只在复合语

句中有效。例如，在例 7.17 主函数的复合语句中定义的变量 c，只在定义它的复合语句内有效，离开复合语句就释放内存单元。

7.9.2 全局变量

在函数外部定义的变量称为全局变量，又称外部变量。全局变量的作用域从定义点开始直到文件尾，可以被作用域内的所有函数共用。

【例 7.18】全局变量举例。

```
int a=1;                  /*定义全局变量 a，作用域从此处到文件结束*/
main()
{ int a=2,b=3;            /*定义局部变量 a、b，只能在主函数 main 中使用*/
  int fun(int x);         /*函数引用说明 */
  printf("a=%d\n",a);     /*输出：a=2 */
  fun(b);                 /*函数调用 */
  printf("a=%d\n",a);     /*输出：a=2 */
}
int c=5;                  /*定义全局变量 c，作用域从此处到文件结束*/
int fun(int x)
{ a+=x+c;
  printf("a=%d\n",a);     /*输出：a=9 */
}
```

运行结果如下：

```
a=2
a=9
a=2
```

说明：

（1）在同一个源文件中，如果全局变量和局部变量同名，则在局部变量的作用域内，全局变量不起作用。例如，在例 7.18 的主函数中定义了局部变量 a，所以全局变量 a 在主函数中不起作用。

（2）全局变量增加了函数之间的联系。由于同一个源文件中的所有函数都能使用全局变量，因此在一个函数中改变了全局变量的值，就能影响到其他函数，相当于各函数间有了直接的数据联系。由于一次函数调用只能带回一个值，因此可以利用全局变量得到一个以上的返回值。

（3）全局变量的使用也存在一些弊端，建议在不必要时不要使用全局变量。首先，使用全局变量会使函数的通用性降低。如果将一个使用全局变量的函数移到另一个文件中，就要将有关的局部变量及其值一起移过去。同时，若该全局变量与其他文件中全局变量同名，就会出现问题，从而降低了程序的可靠性和通用性。其次，使用全局变量增加了程序调试的难度。因为全局变量为多个函数共用，当一个函数出错或被修改时，就有可能影响到其他函数，从而降低了程序的可修改性，增加调试难度。再次，全局变量使用过多还会降低程序的清晰性。因为各函数在执行时都可能改变全局变量的值，因此当某函数执行时，难以判断它使用全局变量的当前值。最后，全局变量在程序的全部执行过程中都占用存储单元，有可能造成存储单元的浪费。

7.10 变量的存储类别

从作用域的角度，变量分为局部变量和全局变量。从生存期（存在时间）的角度，变量又分为静态存储和动态存储。静态存储的变量在整个程序运行期间分配固定的存储空间。动态存储的

变量在程序运行期间根据需要动态分配存储空间，函数调用开始为其分配地址空间，函数调用结束后释放所占空间。

内存中，供 C 程序使用的存储空间分为程序区、静态存储区和动态存储区三部分。程序区专门用于存放源程序（包括函数）；静态存储区用于存放静态型变量，这些变量是在程序编译阶段就已被分配地址并一次进行初始化了，以后不再进行变量初始化工作；动态存储区用于存放动态型变量，这些变量是在函数调用阶段进行地址分配的，函数调用结束后将自动释放其所占内存空间。

C 语言中，变量有数据类型和存储类别两个属性。严格地说，变量的定义形式为：

变量的存储类别 变量的类型 变量名;

读者已经熟知数据类型，如整型、实型、字符型等。存储类别具体分为自动型（auto）、寄存器型（register）、静态型（static）和外部型四种。下面分别加以介绍。

7.10.1　局部变量的存储类别

1．自动型局部变量

自动型局部变量在动态存储区分配存储空间，在调用函数时，系统给它们分配存储空间，在函数调用结束后就自动释放这些存储空间。自动型局部变量用关键字 auto 做存储类别说明，例如：

```
auto float m;
```

在定义局部变量时，若省略存储类别关键字，则系统默认的存储类别为 auto。例如：

```
int a,b,c=10;
```

等价于下面的定义:

```
auto int a,b,c=10;
```

我们前面介绍的在函数中定义的变量都没有声明为 auto，都隐含指定为自动型变量。

2．静态型局部变量

静态型局部变量在静态存储区分配存储空间,变量的值在函数调用结束后不消失而保留原值，即其占用的存储单元不释放，在下一次函数调用时，该变量已有值，就是上一次函数调用结束时的值。静态型局部变量用关键字 static 做存储类别说明，例如：

```
static int n;
```

【例 7.19】静态局部变量举例。

```
fun(int a)                /*a 为形式参数，自动型局部变量*/
{ auto int b=0;           /*b 是自动型局部变量*/
  static int c=3;         /*c 是静态型局部变量，初始化仅进行一次*/
  b+=1;
  c=c+1;
  return a+b+c;
}
main()
{ int a=2,i;              /*a 和 i 都是自动型局部变量*/
  for(i=0;i<3;i++)
    printf("%d\t",fun(a));
}
```

运行结果如下：

```
7   8   9
```

由于静态局部变量在函数调用结束后还保留其值，并对函数下一次调用有影响，所以在使用静态局部变量时一定要慎重。在以下情况下使用静态局部变量。

（1）需要保留上次函数调用结束时的值。

（2）初始化后，变量只被引用而不改变其值。

【例 7.20】打印 1～5 的阶乘值。

```
int fac(int n)
{  static int s=1;          /*s 为静态局部变量，初始化仅进行一次*/
   s*=n;
   return s;
}
main()
{  int i;
   for(i=1;i<=5;i++)
      printf("%d!=%d\n",i,fac(i));
}
```

运行结果如下：

```
1!=1
2!=2
3!=6
4!=24
5!=120
```

说明：

（1）静态型局部变量属于静态存储类别，在静态存储区分配存储单元，在整个程序运行期间不释放，但不能被其他函数引用。自动型局部变量属于动态存储类别，在动态存储区分配存储单元，函数调用结束后即释放。

（2）静态型局部变量在编译时赋初值，在程序运行时已有初值，以后每次调用函数时不再重新赋初值，只是保留上次调用结束时的值。自动型局部变量在函数调用时赋初值，每调用一次函数都重新分配存储单元并赋初值，相当于执行一条赋值语句。

（3）如果在定义静态型局部变量时没有赋初值，编译程序自动对静态型局部变量赋初值，数值型变量为 0，字符型变量为空字符。如果在定义动态型局部变量时没有赋初值，它的值是不确定的。只是因为每次函数调用时都重新分配存储单元，而所分配的存储单元的值是不确定的。

3．寄存器型局部变量

为了提高程序的执行效率，C 语言允许将局部变量的值放在 CPU 的通用寄存器中，这种变量称为寄存器型局部变量。寄存器型局部变量用关键字 register 做存储类型说明，例如：

```
register int a,b;
```

说明：

（1）只有自动型局部变量和形式参数可以说明为寄存器型变量。

（2）由于一个计算机系统中寄存器个数是有限的，所以不能定义任意多个寄存器型局部变量。

在 Turbo C 中，把寄存器型局部变量作为自动型局部变量处理，分配存储单元，并不把它们真正放在寄存器中。

7.10.2　全局变量的存储类别

全局变量是在函数的外部定义的，编译时分配在静态存储区，在整个程序运行期间都占有存储空间。全局变量的作用域为从变量的定义点开始，到它所在的程序文件末尾。通过引用声明可以扩展全局变量的作用域，引用声明的形式为：

extern 变量的类型 变量名；

从作用域的角度来看，全局变量分为外部型和静态型两种。

1．静态型全局变量

在定义全局变量时，若在类型名前加一个关键字 static，则说明定义的变量为静态型全局变量。通过引用声明，可以扩展静态型全局变量的作用域，但只能在它所在的文件中扩展，不能扩展到程序中的其他文件。也就是说，静态型全局变量只能被它所在文件中的函数使用，不能被其他文件中的函数使用。如果一个函数要使用在它后面定义的全局变量，则应该在使用之前作引用声明。这样就将该全局变量的作用域的起始点从定义点扩展到声明处。

【例 7.21】变量的作用域限定在一个文件中。有一个一维数组，存放了若干个学生的成绩。写一函数，计算出平均分、最高分和最低分。

```
#define N 10
static float max,min;           /*定义静态全局变量,只能在本文件内部引用*/
float average(float array[],int n)
{ int i;
  float ave=0.0;
  max=min= array[0];
  for(i=0;i<n;i++)
  { ave+=array[i];
    if(max<array[i]) max=array[i];
    if(min>array[i]) min=array[i];
  }
  ave/=n;
  return ave;
}
main()
{ int j;
  float a[N],ave;
  for(j=0;j<N;j++) scanf("%f",a+j);
  ave= average(a,N);
  printf("ave=%f, max=%f, min=%f\n",ave,max,min);
}
```

运行结果如下：

```
34 12 34 656 4 34 346 54 7 90↙
ave=127.099998，max=656.000000，min=4.000000
```

在本程序中，extern float max,min 是全局变量 max 和 min 的引用说明，其中，变量 max 和 min 的定义是在后面的语句 static float max,min 中进行的。如果不在前面加上 extern float max,min; 程序将会出错。当然，可以将主函数前面的 static float max,min; 移到前面并取代 extern float max,min;。由于 max 和 min 被定义成静态全局变量，所以不能在同一个程序的其他文件中引用这两个变量。

2．外部型全局变量

在定义全局变量时，若没有给出存储类别，则定义的变量为外部型全局变量。我们前面介绍的在函数外定义的未加存储类别的变量都是外部型全局变量。

C 程序由 C 源程序文件组成，C 源程序文件又由函数组成。通过引用声明，外部型全局变量的作用域可以扩展到定义它之前的函数，也可以扩展到程序中的其他文件。也就是说，外部型全局变量不但能被它所在文件中的函数使用，也能被其他文件中的函数使用。如果一个文件要使用另一个文件中定义的外部型全局变量，则在使用它们的文件中作引用声明，说明它们是在其他文件中定义的外部型全局变量。这样就可以在该文件中使用其他文件中定义的外部型全局变量。

【例 7.22】变量的作用域扩展到一个程序的多个文件中。

我们还是完成上面的题目，将上面的程序分为两个文件保存。

文件 M1.C 为：

```
#define N 10
float max,min;                  /*定义了两个全局变量 max 和 min*/
main()
{ int j;
  float a[N],ave;
  extern float average();       /*外部函数引用说明*/
  for(j=0;j<N;j++) scanf("%f",a+j);
  ave= average(a,N);
  printf("ave=%f, max=%f, min=%f\n",ave,max,min);
}
```

文件 S1.C 为：

```
extern float max,min;           /*全局变量引用说明,不在同一个文件中*/
float average(float array[],int n)
{ int i;
  float ave=0.0;
  max=min= array[0];
  for(i=0;i<n;i++)
  { ave+=array[i];
    if(max<array[i]) max=array[i];
    if(min>array[i]) min=array[i];
  }
  ave/=n;
  return ave;
}
```

工程文件 MS1.PRJ 为：

```
M1.C
```

S1.C

运行结果如下：

同例 7.21。

在主函数（main()）中有关于辅助函数（average()）的说明：extern float average();。

在 S1.C 中有变量 max 和 min 的引用说明：extern float max,min;。其中，max 和 min 的定义是在 M1.C 文件中。

本程序运行时需要使用工程文件。工程文件扩展名是.PRJ，它的文本内容由程序中要连接运行的各个源程序文件名一行一行地组成。做好工程文件后，只要把该工程文件名加入到 Turbo C 菜单项 Project 中，程序即可编译连接运行。

7.11　内部函数和外部函数

根据函数能否被其他源文件调用，函数可分为内部函数和外部函数两类。

7.11.1　内部函数

在定义函数时，如果在函数首部的最左端加关键字 static，则表示此函数是内部函数。例如：

```
static int fun(int a,int b)
{ return (a>b?a:b);}
```

内部函数又称静态函数，只能被它所在文件中的函数调用，不能被其他文件中的函数调用。不同文件中的内部函数可以同名，互不干扰。

7.11.2　外部函数

在定义函数时，如果在函数首部的最左端加关键字 extern 或省略关键字，则表示此函数是外部函数。例如：

```
extern int fun(int a,int b)
{ return (a>b?a:b);}
```

内部函数既能被它所在文件中的函数调用，也能被其他文件中的函数调用。C 语言规定，如果在定义函数时省略 extern，则隐含为外部函数。我们前面定义的函数都是外部函数。如果一个文件要调用另一个文件中定义的函数，在需要调用函数的文件中，用 extern 对所调用的外部函数做引用声明。

【例 7.23】通过调用外部函数将一个字符串中的大写字母变为小写。

```
file1.c(文件 1)
main()
{ extern shuru(char str[]);                  /*对外部函数 shuru 做引用声明*/
  extern daxietoxiaoxie(char str[]);         /*对外部函数 shanchu 做引用声明*/
  extern shuchu(char str[]);                 /*对外部函数 shuchu 做引用声明*/
  char str[80];
  shuru(str);
  daxietoxiaoxie (str);
  shuchu(str);
}
```

```
file2.c(文件 2)
#include "stdio.h"
shuru(char str[])                          /*定义外部函数 shuru*/
{ gets(str);}
  file3.c(文件 3)
  daxietoxiaoxie(char str[])               /*定义外部函数 daxietoxiaoxie*/
{ int i;
  for(i=0;str[i];i++)
    if(str[i]>='A'&&str[i]<='Z')
      str[i]+=32;
}
file4.c(文件 4)
shuchu(char str[])                         /*定义外部函数 shuchu*/
{ printf("%s\n",str);}
```

运行结果如下：

```
Ab1%Cdf↙
ab1%cdf
```

7.12 程序设计举例

【例 7.24】编制一个发声程序，当输入命令名和发声数后，发出连续的“嘟嘟”声。

```
#include "stdio.h"
main(int argc,char *argv[])
{ int i,n=0,len;
  len=strlen(argv[1]);       /*计算所给声音数的数字位数*/
  for(i=0;i<len;i++)
  n=10*n+argv[1][i]-'0';     /*将给定的数字串转换成具体的整数*/
  for(i=0;i<n;i++)
  { printf("do");            /*输出单词 do*/
    putchar('\7');           /*发出声音“嘟”*/
    delay(1000);             /*延时 1000*/
  }
}
```

运行结果如下：

假设程序文件名为 Phonec.c，经过编译连接生成了目标程序 Phonec.exe。在 DOS 方式下输入如下命令，则会听到连续的“嘟嘟”声音。

```
Phonec 50↙
"do do do …"
```

【例 7.25】用直接插入排序的方法将一组整数降序排列。

```
insertsort(int *a,int n)
{ int i,j,t;
  for(i=1;i<n;i++)
  { for(t=a[i],j=i-1;j>=0&&t>a[j];j--)
        a[j+1]=a[j];
```

```
        a[j+1]=t;
      }
    }
    #define N 10
    main()
    { int x[N],i;
      for(i=0;i<N;i++)
          scanf("%d",x+i);
      insertsort(x,N);
      for(i=0;i<N;i++)
          printf("%5d",x[i]);
      printf("\n");
    }
```

运行结果如下：

```
34 78 12 90 39 87 47 92 65 17↙
92  90  87  78  65  47  39  34  17  12
```

【例 7.26】汉诺塔问题。

有三根柱子，分别标记为 A、B 和 C。A 柱上有若干个大小不同的盘子叠放成塔形，现将 A 柱上的这些盘子借助于 B 柱移到 C 柱上去。但是，每次只能移动一个盘子，而且大盘不能放到小盘之上。请给出移动方法。

这是一个比较典型的递归程序设计，我们可以想象先把 A 上的除最下面的盘子外都移到 B 柱上，然后把最大的盘子移到 C 柱上，再把 B 柱上的盘子移到 C 柱上，这就可以完成了。但是，把一些盘子移到 B 柱上及从 B 柱上再移到 C 柱上，这本身又是一个递归问题。

```
move(char x,char y)
{
   printf("%c ——>> %c\n",x,y);
}
int count=0;
hanoi(int n,char A,char B,char C)
{
  if(n==1) { move(A,C); count++;}
  else
  { hanoi(n-1,A,C,B);
     move(A,C);count++;
hanoi(n-1,B,A,C);
  }
}
main()
{ int n;
  printf("How many plates? "); scanf("%d",&n);
  hanoi(n,'A','B','C');
  printf("Total moved:%d\n",count);
}
```

运行结果如下：

```
How many plates? 4↙
A ——>> B
A ——>> C
B ——>> C
A ——>> B
C ——>> A
C ——>> B
A ——>> B
A ——>> C
B ——>> C
B ——>> A
C ——>> A
B ——>> C
A ——>> B
A ——>> C
B ——>> C
Total moved:15
```

【例 7.27】将一个 M×N 的二维数组按列优先转存为一个长度为 M×N 的一维数组。

```
#define M 3
#define N 4
void funab(int (*a)[N],int *b,int m,int n) /*a 为数组指针变量名*/
{ int i,j,k=0;
  for(j=0;j<n;j++)                          /*在二维数组中按列扫描数组元素*/
    for(i=0;i<m;i++)
      b[k++]=*(*(a+i)+j);
}
main()
{ int x[M][N],y[M*N];
  int i,j;
  printf("Input 2_dimension array:\n");
  for(i=0;i<M;i++)
    for(j=0;j<N;j++)
      scanf("%d",x[i]+j);
  funab(x,y,M,N);
  printf("The 1_dimension array is:\n");
  for(i=0;i<M*N;i++)
    printf("%d",y[i]);
  printf("\n");
}
```

运行结果如下：

```
Input 2_dimension array:
1 2 3 4↙
5 6 7 8↙
9 10 11 12↙
The 1_dimension array is:
1  5  9  2  6  10  3  7  11  4  8  12
```

【例 7.28】将一个 n 阶方阵的下三角阵按行序存入一个一维数组中。

```
#define N 5
void funab(int a[][N],int *b,int n)
{ int i,j,k=0;
  for(i=0;i<n;i++)
    for(j=0;j<=i;j++)
      b[k++]=*(*(a+i)+j);
}
main()
{ int x[N][N]={{1, 2, 3, 4, 5 },
               {6, 7, 8, 9, 10},
               {11,12,13,14,15},
               {16,17,18,19,20},
               {21,22,23,24,25}},y[N*(N+1)/2];
  int i,j;
  printf("The 2_dimension array is:\n");
  for(i=0;i<N;i++)
  { for(j=0;j<N;j++)
      printf("%4d",x[i][j]);
    printf("\n");
  }
  funab(x,y,N);
  printf("The lower triangular array is:\n");
  for(i=0;i<N;i++)
  { for(j=0;j<=i;j++)
      printf("%4d",y[i*(i+1)/2+j]);
    printf("\n");
  }
}
```

运行结果如下：

```
The 2_dimension array is:
   1   2   3   4   5
   6   7   8   9  10
  11  12  13  14  15
  16  17  18  19  20
  21  22  23  24  25
The lower triangular array is:
   1
   6   7
  11  12  13
  16  17  18  19
  21  22  23  24  25
```

【例 7.29】已知一个长度为 N×(N+1)/2 的一维数组，将其中元素按行序作为下三角阵，构成一个关于主对角线对称的 N 阶方阵。

```
#define N 5
```

```
void funab(int a[],int b[][N])
{ int i,j,k=0;
  for(i=0;i<N;i++)
  {
    for(j=0;j<i;j++)
      b[i][j]=b[j][i]=a[k++];
    b[i][i]=a[k++];
  }
}
main()
{ int x[N*(N+1)/2]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15},y[N][N];
  int i,j,k;
  printf("The 1_dimension as lower triangular array is:\n");
  for(i=k=0;i<N;i++)
  {  for(j=0;j<=i;j++)
       printf("%4d",x[k++]);
     printf("\n");
  }
  funab(x,y);
  printf("The 2_dimension equally triangulars array is:\n");
  for(i=0;i<N;i++)
  { for(j=0;j<N;j++)
      printf("%4d",y[i][j]);
    printf("\n");
  }
}
```

运行结果如下：

```
The 1_dimension as lower triangular array is:
   1
   2   3
   4   5   6
   7   8   9  10
  11  12  13  14  15
The 2_dimension equally triangulars array is:
   1   2   4   7  11
   2   3   5   8  12
   4   5   6   9  13
   7   8   9  10  14
  11  12  13  14  15
```

【例 7.30】将一个字符串中在另一个字符串中出现的字符删除。

```
main()
{ void squ(char a[],char b[]);
  char s1[20]="I am a boy",s2[20]="you are a boy";
  squ(s1,s2);
  printf("\n%s",s1);
}
```

```
void squ(char x[],char y[])
{ int i=0,j=0;
  while(x[i]!='\0')
  { while(y[j]!='\0')
    { if(x[i]==y[j])
      { for(j=i;x[j]=x[j+1];j++);
            i--;
            break;
      }
      j++;
    }
    i++;j=0;
  }
}
```

运行结果如下：

```
Im
```

【例 7.31】求 1！～20！之和。

```
double sum(int n)
{ double s=0,t=1;
  int i;
  for(i=1;i<=n;i++)
  { t*=i;
    s+=t;
  }
  return s;
}
main()
{ double m;
  m=sum(20);
  printf("%f\n",m);
}
```

运行结果如下：

```
2561327494111820290.000000
```

这一结果并不正确，因为 double 型数据的有效位数为 15～16 位，该结果已经超出了这个位数的限制。正确结果应该是：2561327494111820313。

【例 7.32】计算 $1+\frac{1}{1+2}+\frac{1}{1+2+3}+\cdots+\frac{1}{1+2+3+\cdots+n}$

```
double funhe(int n)
{ int i;double t=0,s=0;
  for(i=1;i<=n;i++)
  { t+=i;
    s+=1/t;
  }
  return s;
```

```
}
main()
{ int n;
  printf("n=");scanf("%d",&n);
  printf("%f\n",funhe(n));
}
```

运行结果如下：

```
n=11↙
1.833333
```

【例 7.33】计算 $\frac{1}{1}-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots+(-1)^{n+1}\times\frac{1}{2n-1}$

```
float funsn(int n)
{ int i,f;float s=0,t;
  for(i=1,f=1;i<=n;i++)
  { t=1.0/(2*i-1);
    s+=f*t;
    f=-f;
  }
  return s;
}
main()
{ int n;
  printf("n="); scanf("%d",&n);
  printf("%f\n",funsn(n));
}
```

运行结果如下：

```
n=11↙
0.808079
```

【例 7.34】求 high 以内的 10 个最大素数之和。

```
int funsu(int high)
{ int sum=0,n=0,i,yes;
  while(high>=2&&n<10)
  { yes=1;
    for(i=2;i<=high/2;i++)
      if(high%i==0)
      { yes=0;break;}
    if(yes)
    { n++;
        printf("%d----%d\n",n,high);
        sum+=high;
    }
    high--;
  }
  return sum;
}
```

```
main()
{ int h;
  printf("high=");
  scanf("%d",&h);
  printf("sum=%d\n",funsu(h));
}
```

运行结果如下：

```
high=100↙
1----97
2----89
3----83
4----79
5----73
6----71
7----67
8----61
9----59
10----53
sum=732
```

本 章 小 结

函数是 C 语言中最重要的概念之一。本章重点介绍了函数的定义、形参和实参，以及函数的调用及其在函数调用过程中参数的传递方式。函数由函数名、函数返回值、参数列表和函数体构成。函数的参数和返回值是函数对外交互的接口。

函数调用是 C 语言复杂程序设计中非常重要的概念。本章详细介绍了函数的嵌套调用和递归调用，同时还列举了大量的实例代码来说明函数调用的原理和方法。

最后介绍了函数及程序设计中，变量的生存期和作用域以及存储类别。函数还可划分为内部函数和外部函数。

习　题　7

一、填空题

1．函数的定义不可以嵌套，但函数的调用_____嵌套。

2．函数调用时的实参和形参之间的数据是单向的_____传递。

3．如果函数不要求带回值，可用_____来定义函数返回值为空。

4．静态变量和外部变量的初始化是在_____阶段完成的，而自动变量的赋值是在_____时进行的。

5．函数的_____调用是一个函数直接或间接地调用它自身。

6．函数调用语句 func((e1,e2),(e3,e4,e5))中含有_____个实参。

二、单选题

1．若使用一维数组名作为函数实参，则以下正确的说法是（　　）。

A．必须在主调函数中说明此数组的大小
B．实参数组类型与形参数组类型可以不匹配
C．在被调用函数中，不需要考虑形参数组的大小
D．实参数组名与形参数组名必须一致

2．在 C 语言中，函数的隐含存储类别是（　　）。
A．auto　　B．static　　C．extern　　D．无存储类别

3．若执行 fopen 函数时发生错误，则函数的返回值是（　　）。
A．地址值　　B．0　　C．1　　D．EOF

4．C 语言规定，函数返回值的类型由（　　）。
A．return 语句中的表达式类型决定
B．调用该函数时的主调函数类型决定
C．调用该函数时系统临时决定
D．在定义该函数时所指定的函数类型决定

5．在 C 语言程序中，（　　）。
A．函数的定义可以嵌套，但函数的调用不可以嵌套
B．函数的定义不可以嵌套，但函数的调用可以嵌套
C．函数的定义和函数的调用均可以嵌套
D．函数的定义和函数的调用不可以嵌套

6．求平方根函数的函数名为（　　）。
A．cos　　B．abs　　C．pow　　D．sqrt

7．C 语言中函数调用的方式有（　　）。
A．函数调用作为语句一种
B．函数调用作为函数表达式一种
C．函数调用作为语句或函数表达式两种
D．函数调用作为语句、函数表达式或函数参数三种

8．下列叙述中正确的是（　　）。
A．C 语言编译时不检查语法　　B．C 语言的子程序有过程和函数两种
C．C 语言的函数可以嵌套定义　　D．C 语言所有函数都是外部函数

9．执行下面程序后，输出结果是（　　）。

```
main()
{ a=45,b=27,c=0;
  c=max(a,b);
  printf("%d\n",c);}
  int  max(x,y)
  int  x,y;
  { int z;
    if(x>y)
  z=x;
else
  z=y;
  return(z);
}
```

A．45　　B．27　　C．18　　D．72

10．在 C 语言中，调用函数除函数名外，还必须有（　　）。

A．函数预说明　　B．实际参数　　C．( )　　D．函数返回值

三、判断题

1．如果函数值的类型和 return 语句中表达式的值不一致，则以函数类型为准。（　　）

2．通过 return 语句，函数可以带回一个或一个以上的返回值。（　　）

3．进行宏定义时，宏名必须使用大写字母表示。（　　）

4．函数 strlen("ASDFG\n")的值是 7。（　　）

5．C 程序中有调用关系的所有函数必须放在同一个源程序文件中。（　　）

四、程序填空题

1．题目：以下程序的功能是求

$$y=\begin{cases} 2x^2+3x+4 & x<2 \\ -2x^2+3x-4 & x>2 \end{cases}$$

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
/***********SPACE**********/
double  f(【?】)
{ /***********SPACE**********/
  【?】;
/***********SPACE**********/
   if (【?】)
     y=2.0*x*x+3.0*x+4.0;
   else
     y=-2.0*x*x+3.0*x-4.0;
/***********SPACE**********/
  【?】}
main()
{ clrscr();
  printf("%f\n", f(-1.9)+f(5.0));}
```

2．题目：给定程序中函数 fun 的功能是，将长整型数中每一位上为奇数的数依次取出，构成一个新数放在 t 中。高位仍在高位，低位仍在低位。

```
#include <conio.h>
#include <stdio.h>
void fun (long s, long *t)
{ int d;
long sl=1;
/***********SPACE**********/
【?】 = 0;
while ( s > 0)
/***********SPACE**********/
{ d = 【?】;
if(d%2)
```

```
/***********SPACE***********/
       { *t = 【?】 + *t;
/***********SPACE***********/
       sl 【?】 10;
       }
s /= 10;
   }
}
main()
{ long s, t;
clrscr();
printf("\nPlease enter s:"); scanf("%ld", &s);
fun(s, &t);
printf("The result is: %ld\n", t);}
```

3．题目：给定程序函数 fun 的功能是，统计子字符串 substr 在字符串 str 中出现的次数。例如，若字符串为 aaas lkaaas，子字符串为 as，则应输出 2。若字符串为 asasasa，子字符串为 asa，则应输出 3。

```
#include <stdio.h>
fun (char *substr,char *str)
{  int i,j,k,num=0;
   for(i=0; str[i]; i++)
     for(j=i,k=0;substr[k]==str[j];k++,j++)
/***********SPACE***********/
      if(substr[【?】]=='\0')
      {  num++;
/***********SPACE***********/
         【?】;  }
   return num;}
main()
{ char str[80],substr[80];
  printf("Input a string:") ;
  gets(str);
  printf("Input a substring:") ;
  gets(substr);
/***********SPACE***********/
  printf("%d\n",【?】); }
```

4．题目：编程求任意给两个日期（Y0 年 M0 月 D0 日和 Y1 年 M1 月 D1 日）相差的天数。

```
main()
{int y1,m1,d1,y2,m2,d2,n,i;
 printf("y1,m1,d1=");scanf("%d,%d,%d",&y1,&m1,&d1);
/***********SPACE***********/
 if(m1<1||m1>12||d1<1||d1>【?】)exit(0);
 printf("y2,m2,d2=");scanf("%d,%d,%d",&y2,&m2,&d2);
/***********SPACE***********/
 if(m2<1||m2>12||d2<1||d2>【?】)exit(0);
 if(y1>y2||y1==y2&&m1>m2||y1==y2&&m1==m2&&d1>d2)
```

```
 {n=y1;y1=y2;y2=n;n=m1;m1=m2;m2=n;n=d1;d1=d2;d2=n;}
 else
/***********SPACE**********/
 {n=yend(y1,m1,d1)+【?】;
/***********SPACE**********/
 for(i=【?】;i<y2;i++)n+=365+f(i);}
 printf("%d.%d.%d--->%d.%d.%d:n=%d\n",y1,m1,d1,y2,m2,d2,n);}
int f(int y)
{return(y%4==0&&y%100!=0||y%400==0);}
int mday(int y,int m)
{return(31-((m==4)+(m==6)+(m==9)+(m==11))-(3-f(y))*(m==2));}
int yday(int y,int m,int d)
{return(d+31*((m>1)+(m>3)+(m>5)+(m>7)+(m>8)+(m>10))+30*((m>4)+(m>6)+(m>9)+
(m>11))+(28+f(y))*(m>2));}
int yend(int y,int m,int d)
{return(365+f(y)-yday(y,m,d));}
```

5. 题目：数组名作为函数参数，求平均成绩。

```
float aver(float a[ ])     /*定义求平均值函数，形参为一浮点型数组名*/
{int i;
float av,s=a[0];
for(i=1;i<5;i++)
/***********SPACE**********/
s+=【?】[i];
av=s/5;
/***********SPACE**********/
return 【?】;}
void main()
{float sco[5],av;
int i;
printf("\ninput 5 scores:\n");
for(i=0;i<5;i++)
/***********SPACE**********/
scanf("%f",【?】);
/***********SPACE**********/
av=aver(【?】);
printf("average score is %5.2f\n",av);
getch();}
```

五、程序改错题（注意：不可以增加或删除程序行，也不可以更改程序的结构。）

1. 题目：利用递归函数调用方式，将所输入的 5 个字符以相反顺序打印出来。

```
#include"stdio.h"
main()
{
int i=5;
void palin(int n);
printf("\40:");
palin(i);
```

```
printf("\n");
}
void palin(n)
int n;
{
/**********FOUND**********/
int next;
if(n<=1)
{
/**********FOUND**********/
next!=getchar();
printf("\n\0:");
putchar(next);
}
else
{
next=getchar();
/**********FOUND**********/
palin(n);
putchar(next);
}
}
```

2．题目：编写程序，求矩阵（3 行 3 列）与 5 的乘积。例如，输入下面的矩阵：

```
100 200 300
400 500 600
700 800 900
```

程序输出：

```
500  1000 1500
2000 2500 3000
3500 4000 4500
```

```
#include "stdio.h"
int fun(int array[3][3])
{
/**********FOUND**********/
int i;j;
/**********FOUND**********/
for(i=1; i < 3; i++)
for(j=0; j < 3; j++)
/**********FOUND**********/
array[i][j]=array[i][j]/5;
}
main()
{int i,j;
int array[3][3]={{100,200,300},{400,500,600},{700,800,900}};
clrscr() ;
for (i=0; i < 3; i++)
```

```
{  for (j=0; j < 3; j++)
   printf("%7d",array[i][j]);
printf("\n");
}
fun(array);
printf("Converted array:\n");
for (i=0; i < 3; i++)
{  for (j=0; j < 3; j++)
   printf("%7d",array[i][j]);
printf("\n");
}
getch();
}
```

3．题目：请编写一个函数 int fun(int x)。它的功能是：判断整数 x 是否是同构数。若是同构数，函数返回 1；否则，返回 0。所谓“同构数”是指这个数出现在它的平方数的右边。例如：输入整数 25，25 的平方数是 625，25 是 625 中右侧的数，所以 25 是同构数。x 的值由主函数从键盘输入，要求不大于 1000。

```
#include "stdio.h"
int fun(int x)
{
/**********FOUND**********/
  int k
/**********FOUND**********/
    k=x;
/**********FOUND**********/
   if((k%10==x)&&(k%100==x)&&(k%1000==x))
   return 1;
  else
   return 0;
}
main()
{
  int x,y;
 clrscr();
  printf("\nPlease enter a integer numbers:");
  scanf("%d",&x);
  if(x>100){printf("data error!\n");exit(0);}
 y=fun(x);
 if(y) printf("%d YES\n",x);
 else printf("%d NO\n",x);
 getch();}
```

4．题目：fun 函数的功能是，给定 n 个实数，输出平均值，并统计在平均值以下（含平均值）的实数个数。例如，n=6 时，输入 23.5，45.67，12.1，6.4，58.9，98.4，所得平均值为 40.828 335，在平均值以下的实数个数应为 3。请改正程序中的错误，使它得出正确的结果。

```
int fun(float x[],int n)
```

```
 {
  int j,c=0;
/**********FOUND**********/
  float j=0;
/**********FOUND**********/
  for(j=0;j<=n;j++)
     xa+=x[j];
     xa=xa/n;
     printf("ave=%f\n",xa);
/**********FOUND**********/
  for(j=0;j<=n;j++)
    if(x[j]<=xa)   c++;
/**********FOUND**********/
  return xa;
}
main()
{
  float x[]={23.5,45.67,12.1,6.4,58.9,98.4};
  printf("%d\n",fun(x,6));
}
```

5．题目：给定程序 MODI1.C 中函数 fun 的功能是，求出数组中最大数和次最大数，并把最大数和 a[0]中的数对调、次最大数和 a[1]中的数对调。

```
#include <conio.h>
#include <stdio.h>
#define N 20
int fun ( int * a, int n )
{ int i,m,t,k;
for(i=0;i<2;i++)
/**********FOUND**********/
{ m=0;
/**********FOUND**********/
for(k=1;k<n;k++)
/**********FOUND**********/
 if(a[k]>a[m]) k=m;
t=a[i];a[i]=a[m];a[m]=t;
}
}
main()
{ int x, b[N]={11,5,12,0,3,6,9,7,10,8}, n=10, i;
clrscr();
for (i=0;i<n;i++) printf("%d", b[i]);
printf("\n");
fun (b,n);
for ( i=0; i<n; i++ ) printf("%d", b[i]);
printf("\n");
}
```

六、程序设计题（注意：部分源程序已给出，请勿改动主函数 main 和其他函数中的任何内容，仅在函数的花括号中填入所编写的若干语句。）

1．题目：用函数求 fibonacci 数列前 28 项的和。已知数列的第一项值为 1，第二项值也为 1，从第三项开始，每一项均为其前面相邻两项的和。运行结果为 832 039。

```
#include "stdio.h"
long sum(long f1,long f2)
{
/**********Program**********/









/**********  End  **********/
  }
main()
{long int f1=1,f2=1;
  clrscr();
  printf("sum=%ld\n",sum(f1,f2));
  yzj();
  getch();
  }
yzj()
{
FILE *IN,*OUT;
int m,n;
  int i[2];
long int o;
IN=fopen("in.dat","r");
if(IN==NULL)
{printf("Read File Error");
}
OUT=fopen("out.dat","w");
if(OUT==NULL)
{printf("Write File Error");
}
for(n=0;n<2;n++)
   fscanf(IN,"%d",&i[n]);
o = sum(i[0],i[1]);
fprintf(OUT,"%ld\n",o);

fclose(IN);
fclose(OUT);
}
```

2．题目：请编一个函数 void fun(int tt[M][N],int pp[N])，tt 指向一个 M 行 N 列的二维数组，

求出二维数组每列中最大元素，并依次放入 pp 所指一维数组中。二维数组中的数已在主函数中赋予。

```
#include<conio.h>
#include<stdio.h>
#define M 3
#define N 4
void fun(int tt[M][N],int pp[N])
{
/**********Program**********/









/**********  End  **********/
}
main()
{
 int t[M][N]={{22,45,56,30},{19,33,45,38},{20,22,66,40}};
  int p[N],i,j,k;
  clrscr();
  printf("The original data is:\n");
  for(i=0;i<M;i++)
  {
    for(j=0;j<N;j++)
       printf("%6d",t[i][j]);
       printf("\n");
   }
   fun(t,p);
   printf("\nThe result is:\n");
   for(k=0;k<N;k++) printf("%4d",p[k]);
   printf("\n");
   getch();
   NONO();
}
NONO()
{
int i,j, array[3][4],p[4];
FILE *rf, *wf ;
rf = fopen("in.dat", "r");
wf = fopen("out.dat", "w");
for (i=0; i < 3; i++)
for (j=0; j < 4; j++)
fscanf(rf, "%d", &array[i][j]);
fun(array,p);
for (j=0; j < 4; j++)
```

```
{
fprintf(wf, "%7d", p[j]);
fprintf(wf, "\n");
}
fclose(rf);
fclose(wf);
}
```

3. 题目：编写程序，实现矩阵（3 行 3 列）的转置（即行列互换）。例如，输入下面的矩阵：

```
100 200 300
400 500 600
700 800 900
```

程序输出：

```
100 400 700
200 500 800
300 600 900
```

```
#include "stdio.h"
int fun(int array[3][3])
{
/**********Program**********/




/**********  End  **********/
}
main()
{
int i,j;
int array[3][3]={{100,200,300},
{400,500,600},
{700,800,900}};
clrscr() ;
for (i=0; i < 3; i++)
{ for (j=0; j < 3; j++)
printf("%7d",array[i][j]);
printf("\n");
}
fun(array);
printf("Converted array:\n");
for (i=0; i < 3; i++)
{ for (j=0; j < 3; j++)
printf("%7d",array[i][j]);
printf("\n");

}
```

```
getch();
NONO();
}
NONO()
{
int i,j, array[3][3];
FILE *rf, *wf ;
rf = fopen("in.dat", "r");
wf = fopen("out.dat", "w");
for (i=0; i < 3; i++)
for (j=0; j < 3; j++)
fscanf(rf, "%d", &array[i][j]);
fun(array);
for (i=0; i < 3; i++)
{ for (j=0; j < 3; j++)
fprintf(wf, "%7d", array[i][j]);
fprintf(wf, "\n");
}
fclose(rf);
fclose(wf);
}
```

4．题目：编写函数 int fun(int lim,int aa[MAX])，该函数的功能是求大于 lim（lim 为小于 100 的整数）并且小于 100 的所有素数并放在 aa 数组中，该函数返回所求出素数的个数。

```
#include<stdio.h>
#include<conio.h>
#define MAX 100
int fun(int lim,int aa[MAX])
{
/**********Program**********/









/**********  End  **********/
}
main()
{
  int limit,i,sum;
  int aa[MAX];
  clrscr();
  printf("Please Input aInteger:");
  scanf("%d",&limit);
  sum=fun(limit,aa);
  for(i=0;i<sum;i++){
```

```
    if(i%10==0&&i!=0) printf("\n");
    printf("%5d",aa[i]);

   }
   NONO();
   getch();
}
NONO()
{
int i,j,array[100],sum,lim;
FILE *rf, *wf;
rf = fopen("in.dat", "r");
wf = fopen("out.dat", "w");
for (j=0; j <= 5; j++)
{
fscanf(rf, "%d", &lim);
sum=fun(lim,array);
for(i=0;i<sum;i++)
fprintf(wf, "%7d", array[i]);
fprintf(wf, "\n");
}
fclose(rf);
fclose(wf);
}
```

5．编写函数 fun，函数的功能是：计算 n 门课程的平均值，计算结果作为函数值返回。例如，若有 5 门课程的成绩是 92，76，69，58，88，则函数的值为 76.6。

```
#include "stdio.h"
float fun(int a[],int n)
{
/**********Program**********/









/**********  End  **********/
}

main()
{
int a[]={92,76,69,58,88};
printf("y=%f\n",fun(a,5));
}
```

第 8 章　指　　针

欢迎走进内存这片雷区。比尔·盖茨曾经说过，640KB 内存对于大多数应用来说应该是足够了。看来，天才也有说错话的时候，内存管理程序往往是最令程序员感到麻烦的地方，也是程序“bug 集中营”。因此，掌握内存的基本知识是十分必要的，本章要介绍内存的使用以及 C 语言的难点——指针。

本章包括的知识点：

- 内存和地址
- 指针的声明和定义
- 如何使用指针
- 指针的运算
- 动态内存分配
- 指针和数组的操作
- 结构体和结构体数组的使用
- 函数指针

8.1　计算机中的内存

熟悉计算机的读者都知道，内存是平时接触较多的一个概念。从硬件上讲，内存是一个物理设备；从功能上讲，内存是一个数据库，程序在执行前都要装载到内存中，才能被中央处理器执行。

以 Windows 系统为例，执行安装在硬盘上的某个程序，实际上是将该程序的指令和数据读入内存，供中央处理器执行的过程。

内存是由按顺序编号的一系列存储单元组成的。在内存中，每个存储单元都有唯一的地址，通过地址可以方便地在内存单元中存取信息。内存中的数据要靠电源来维持，当计算机关机或意外断电时，其中的所有数据就永久地消失了。

8.1.1　内存地址

可以将内存看成一个个连续的小格子的集合，为了正确地访问这些小格子，必须给这些小格子编号，正如我们平时讲某栋房屋在 A 小区 X 楼 Y 单元 Z 房间一样，这个 A、X、Y 和 Z 等实际上是对该房间的编号，有了这个编号，或者更通俗地说是“地址”，我们就能从一个城市的成千上万栋几乎一样的房子中找到该房间。

内存地址的引入是同样的道理，为了正确地访问每个内存单元，要对其进行编址，以 32 位计算机为例，其地址空间为 32 位，采用 32 位地址编码，如 0X87654321 的形式。

内存地址是连续的，相邻内存单元间的地址差 1，可以把内存看成一个平坦连续的一维空间。

8.1.2　内存中保存的内容

在计算机中，一切信息都是以二进制数据的形式存放的，每个内存单元的容量是 1B，即 8bit（8 个二进制数）。

中央处理器（CPU）进行的所有运算都离不开内存。使用过 Windows 系统的读者都知道，双击某个可执行程序，CPU 会执行它，这实际上是复杂的内存载入过程。

（1）程序所要进行操作的对应代码装载到代码区。

（2）全局的静态数据等装载到数据区。

（3）开辟堆栈，供临时变量等使用。

可见，内存中的数据是多种多样的，既可以是程序，也可以是数据，都存储在一个个的内存小格子中，每个小格子存储 8 个二进制数。

8.1.3　地址就是指针

在说明指针概念之前，本节将使读者对指针有个感性的认识。所谓指针，指的是“储存的内容是地址的量”，这个概念包括两个要点：

（1）指针是个量，对应着一块内存区域。

（2）指针存储的信息是某个内存单元的地址。

这类似于现实生活中的地址和名片，习惯于把地址印在名片上，这里名片的作用和指针相仿，存储的都是地址数据。

8.2　指针的定义

本节将解释一个问题：如何定义一个指针。指针是 C 语言管理内存的强大工具。

8.2.1　指针变量的声明

指针可以视为一个普通变量，通常所说的定义一个指针实际上是声明一个指针变量的过程，编译器根据指针变量声明语句，为指针变量开辟内存空间，使其有实际意义，这样指针变量才可用。

在声明一个指针变量时，需要向编译器提供以下信息：

- 指针的类型，原则上，指针类型应与其指向的数据类型一致，但也有例外，稍后介绍。
- 指针变量名。

举例来说，下述语句用于声明一个指向 int 型数据的指针 pInt：

```
int* pInt;
```

不难看出，要声明一个指向某种类型的指针变量，其基本形式为：

```
类型* 指针变量名;
```

要在一行语句中同时声明两个指针变量，后面的指针变量同样要加星号，例如：

```
int* p1=NULL, *p2=null;
```

NULL 是 C 语言中预定义的空指针关键字。

8.2.2　指针变量的初始化

在声明一个指针后，编译器并不会自动完成其初始化。此时，指针的值是不确定的，也就是说，该指针的值取决于指针所占内存区域的值，而该值是完全随机的。因此，指针变量的初始化十分重要，直接使用未加初始化的指针变量，可能会给程序带来各种内存错误，因为完全不知道指针指向的是哪块内存，通过指针操作的又是哪块内存。

如果在指针变量声明之初确实不知道应该将此指针指向何处，最简单的方式是将其置为“0”，C 语言中提供了关键字 NULL，如下：

```
int* pInt=NULL;
```

这样，指针 pInt 便不会在内存中乱指一气。

如果要让指针变量确切地指向某个变量，需要使用取地址操作符&。

8.2.3 指针变量的值

“指针变量的值”是指针本身存储的数值，这个值将被编译器作为一个地址，而不是一个一般的数值。在 32 位程序里，所有类型的指针的值都是一个 32 位整数，因为 32 位程序里内存地址的长度都为 32 位。“指针所指向的内存区”就是从指针的值所代表的那个内存地址开始，长度为 sizeof（指针所指向的类型）的一片内存区。

请注意，在本书中，说“一个指针的值是 A”，是指“该指针指向了以 A 为首地址的一片内存区域”；反之，说“一个指针指向了某内存区域”，是指“该指针的值是这块内存区域的首地址”。

8.2.4 取地址操作符&

声明一个变量时，为该变量开辟内存空间的任务是由编译器自动完成的，用户不必关心变量在内存中的位置。但是，如果程序中用到某个变量的地址信息，则该怎么办呢？

C 语言提供了取地址操作符&返回某程序实体的地址信息，举例来说：

```
int  num=0:;
int*  p=&num;
```

&num 返回的是变量 num 在内存中的地址信息，可以直接将此地址赋值给同类型的指针 P。

8.2.5 指针变量占据一定的内存空间

指针变量声明后，编译器为其开辟一定的内存空间，即指针变量占据一定的内存空间，而且，不论是何种类型的指针，都占据 4 个内存字节（这是由 32 位地址数据决定的）。

【例 8.1】指针变量占据内存空间的大小 SizeOfPointer。

文件名：SizeOfPointer.c

```
01 #include  <stdio.h>  /*使用 printf 要包含的头文件*/
02 #include  <conio.h>
03 main()               /*主函数*/
04 {
05   int  Inum=0;        /*声明 int 型变量 Inum,初始化为 0*/
06   short   Snum=0;     /*声明 short 型变量 Snum,初始化为 0*/
07   double   Dnum=0;    /*声明 double 型变量 Dnum,初始化为 0*/
08   int*  pInt=&Inum;   /*声明 int 型指针变量 pInt,用 Inum 地址为其初始化*/
09   short*pShort=&Snum; /*声明 short 型指针变量 pShort,用 Snum 地址为其初始化*/
10  double*pDouble=&Dnum;/*声明 Double 型指针变量 pDouble,用 Dnum 地址初始化*/
11  /*输出 int 型变量 Inum 占据的内存字节数*/
12  printf("Inum 占据的内存字节数为:%d\n",sizeof(Inum));
13  /*输出 int 型指针变量 pint 占据的内存字节数*/
14  printf("pInt 占据的内存字节数为:%d\n",sizeof(pInt));
15  /*输出 short 型变量 Snum 占据的内存字节数*/
```

```
16  printf("Snum 占据的内存字节数为:%d\n",sizeof(Snum));
17  /*输出 short 型指针变量 pShort 占据的内存字节数*/
18  printf("pShort 占据的内存字节数为:%d\n",sizeof(pShort));
19  /*输出 double 型变量 Dnum 占据的内存字节数*/
20  printf("Dnum 占据的内存字节数为:%d\n",sizeof(Dnum));
21  /*输出 double 型变量 pDouble 占据的内存字节数*/
22  printf("pDouble 占据的内存字节数为:%d\n",sizeof(pDouble));
23  getch();     /*等待,按任意键继续*/
24  }
```

运行结果如下：

```
Inum 占据的内存字节数为：4
pInt 占据的内存字节数为：4
Snum 占据的内存字节数为：2
pShort 占据的内存字节数为：4
Dnum 占据的内存字节数为：8
pDouble 占据的内存字节数为：4
```

分析：从例 8.1 中不难看出，不论变量是何种类型，占据多大的内存空间，指针变量占据的内存字节数恒为 4。由此可见，内存变量占据的空间和其指向的变量占据的空间是两码事。

注意：指针变量的值与指针变量的地址是有区别的。

8.2.6　指向指针的指针

指针变量也是变量，占据一定的内存空间，有地址。因此，可以用一个指针指向它，这称为指向指针的指针或二级指针，如图 8.1 所示。

可以通过“**”声明一个二级指针，例如：

```
double num;
double* pN=&num;
double** ppN=&pN;
```

图 8-1　指向指针的指针

上面的指针可以看成指向 double*变量类型的指针。若有需要，还可以定义三级甚至更多的指针。

8.3　使 用 指 针

正如拿着名片可以找到某个人一样，通过指针可以访问其指向的某块内存区域。为此，C 语言引入了间接引用的概念，这需要使用运算符*。

8.3.1　运算符*

上节在讨论指针变量声明时已经说明了*的作用，可以将“类型”视为一个整体，结合类型，用其可以声明某种类型的变量。除此之外，*的另一个作用是“间接引用”，则通过指针访问其指向的内存区域。

【例 8.2】使用指针间接访问其指向的变量 DeferenceSample。

文件名：DeferenceSample.c

```
01  #include <stdio.h>      /*使用 printf 要包含的头文件*/
```

```
02  #include <conio.h>
03  main()                    /*主函数*/
04  {
05  int num=9;
06  /*声明 int 型指针变量 pInt,用 num 地址为其初始化*/
07  int* pInt=&num;
08  /*以十六进制形式输出 pInt 的值,即 num 的内存地址*/
09  printf("指针变量 pInt 的值为:%x\n",pInt);
10  /*以十进制形式输出 pInt 的值,即 num 的内存地址*/
11  printf("指针变量 pInt 指向的内存区域为:%d\n,",pInt);
12  *pInt=10;                          /*通过指针改写其指向的内存区域*/
13  printf("num 变量的值为:%d\n",num);   /*输出 num 的值*/
14  getch();                           /*等待,按任意键继续*/
15  }
```

运行结果如下：

```
指针变量 pInt 的值为:12FF68
指针变量 pInt 指向内存区域为: 9
num 的值为:10
```

分析：由例 8.2 不难看出，指针变量 pInt 的值为 0012FF68，这实际上是 int 型 num 在内存中的地址，在使用语句“int num=9;”声明 int 型变量 num 时，编译器自动为 num 开辟内存空间，这块内存空间的首地址是 0012FF68，而后，编译器将这块 int 型区域初始化为 9。而使用“指针”的形式可间接访问指针所指的内存空间，换言之，在例 8.2 中，*pInt 等价于 num，同时，通过间接引用*pInt=10;可改写指针指向的区域。

对前面提到的概念做一下辨析和总结，请看下列代码：

```
double num=3;
double *pNum;
pNum=&num;
```

对以上代码说明如下：

- num：double 类型的变量。
- pNum：指向 double 类型的指针变量，其值是 num 的地址。
- &num：返回变量 num 的地址，与 pNum 等价。
- *pNum：pNum 所指的变量，间接访问方式，与 num 等价。
- &（*pNum）：与&num（即 pNum）等价，num 的地址。
- *（&num）：与*pNum（即 num）等价，变量 num。

8.3.2 指针的类型和指针所指向的类型

原则上说，指针的类型和指针所指向的类型应当是相同的，但也有例外。讨论之前，区分一下两个概念：所谓指针的类型，指的是声明指针变量是位于变量名前的“类型*”；而所谓指针所指向的类型，指的是为指针初始化或赋值的变量类型。

理解两个类型的不同是掌握 C 语言指针的关键所在。本节从以下两个方面讲述，让读者体会两者的不同。

（1）指针的类型和指针所指向的类型相同时，指针的赋值。

（2）指针的类型和指针所指向的类型不同时，指针的赋值。

```
p1=p2;                                  /*指针间相互赋值*/
```

或

```
p1=&num;                                /*取变量地址给指针赋值*/
```

8.3.3　同类型指针的赋值

同类型指针的赋值是最常见的一种情况。例如，pN1 和 pN2 是两个相同类型的指针，执行“pN2=pN1;”这样一个赋值操作后，pN1 和 pN2 指向同样的地址，也就是说，两个指针指向同一个内存单元，对*pN2 的任何改动都会影响*pN1 的值，反之亦然。

【例 8.3】同类型指针赋值 SameTypePtr。

```
文件名：SameTypePtr.c
01  #include <stdio.h>      /*使用 printf 要包括的头文件*/
02  #include <conio.h>
03  main()                  /*主函数*/
04  {
05    int num=9;            /*声明 int 型变量 num,初始化 9*/
06    /*声明 int 型指针变量 pIntl,用 num 地址为其初始化*/
07    int* pIntl=&num;
08    *pIntl=10;            /*通过指针改写其指向的内存区域*/
09    printf("num 变量的值为:%d\n",num);   /*输出 num 的值*/
10    int*pInt2=pInt1;
11   *pInt2=11;             /*等价于*pInt1=11:;*/
12    printf((num 变量的值为:%d/n",num);
13    getch();              /*等待,按任意键继续*/
14  }
```

运行结果如下：

```
num 变量的值为：10
num 变量的值为：11
```

分析：由例 8.3 不难看出，同类型指针赋值后，*pIntl、*pInt2 和 num 实质上是等价的。

8.3.4　指针的类型和指针所指向的类型不同

“指针的类型和指针所指向的类型不同”是指如下情况。

（1）指针内存的字节数大于指针类型占据的字节数。

```
double Dnum;
int  *pI =&Dnum;          /*pI 为 int 型指针，而 Dnum 是 double 型变量*/
```

或

```
double Dnum;
double *p2=&Dnum;
int *p1=p2;               /*p1 为 int 型指针，而 p2 为 double 型指针*/
```

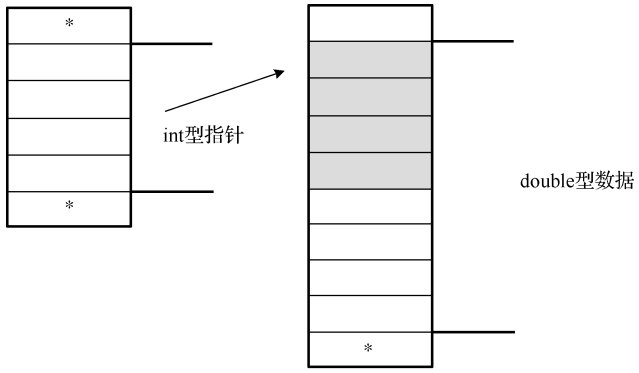
（2）指向内存的字节数小于指针类型占据的字节数。

```
short  Snum;
double  * pD=&Snum;      /*pD 为 double 型指针，而 Snum 是 short 型变量*/
```

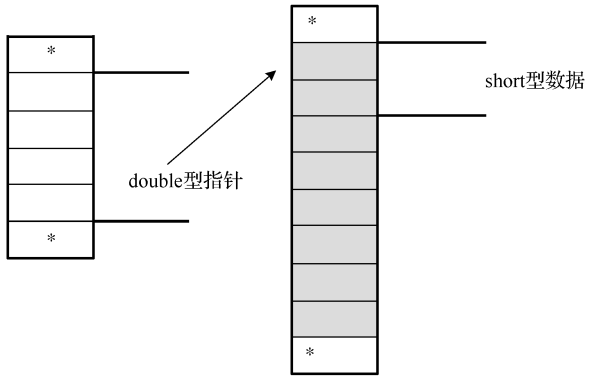
或

```
short  Snum;
short  *p2=&Snum;
double  *p1=p2;          /*p1 为 double 型指针，而 p2 是 short 型指针*/
```

根据前面讨论过的“赋值数据转换”，编译器并不会指明上述语句有错，但大都会给出警告信息，提示用户注意。此时，只是简单地将变量在内存中的首地址赋值给指针，如图 8-2 所示，阴影部分代表可通过指针间接访问的区域，可见，其只取决于指针的类型，编译器并不关心原来内存处是何种类型的数据，对于图(a)的情况，尽管 double 型变量占据 8 个内存字节，但使用指针只能管理前 4 个字节；对于图(b)的情况，short 型数据占据两个内存字节，而 double 型指针管理 8 个字节。这时，short 型数据后面的 6 个字节便会被该指针改写，这种“越权”往往会给程序带来致命的后果。试想，如果 short 型数据后的内存单元很重要的数据被无意修改了，程序肯定是要出错的。



(a) 指向内存的字节数大于指针类型占据的字节数



(b) 指向内存的字节数小于指针类型占据的字节数

图 8-2　指针的类型和指针所指的类型不同示意图

使用指针间接访问某块内存时，编译器根据指针的类型解释内存信息（二进制序列），由于不同类型的数据在内存中的表示形式不同，同样的二进制串会出现不同的解释形式，因此，如图 8-2 所示的赋值是没有实际意义的。

（3）有时，两个不同的类型可能占据相同的内存字节数。此时，内存数目上不会出现出入，但问题还是出在数据的表示形式上，见例 8.4。

【例 8.4】不同类型指针赋值 DiffTypePtr。

```
文件名：DiffTypeptr.c
01  #include <stdio.h>          /*使用 printf 要包含的头文件*/
02  #include <conio.h>          /*主函数*/
03  main()
04  {
05      long num=9;             /*声明 int 型变量 num，初始化为 9*/
06  /*声明 int 型指针变量 pInt，用 Inum 地址为 其初始化*/
07      float* pF=&num;
08      printf("num 变量的值为:%d\n",num);     /*输出 num 的值*/
09      printf("*pF 变量的值为:%f\n",*pF);     /*输出的 num 的值*/
10      *pF=5.0;
11       printf("num 变量的值为:%d\n",num);    /*输出 num 的值*/
12       printf("*pF 变量的值为:%f\n",*pF);    /*输出 num 的值*/
13       getch();                              /*等待，按任意键继续*/
14  }
```

运行结果如下：

```
num 变量的值为：9
*pF 变量的值为：0.000000
num 变量的值为：1084227584
*pF 变量的值为：5.000000
```

分析：虽然 long 型数据和 float 数据都占据 4 个内存字节，但由于两种类型在内存中的表示形式差异巨大，编译器对二进制位的解析也有很大的不同，因此，例 8.4 输出了看似奇怪的结果。

8.4　指针的运算

作为一种特殊的变量，指针可以进行一些运算，但并非所有的运算都是合法的，指针的运算主要局限在加减算数运算和其他一些为数不多的特殊运算上。

8.4.1　算术运算之“指针+整数”或者“指针–整数”

指针和整数的加减返回结果也是一个指针，确切地说是个地址值，问题的关键在于这个指针到底指向什么地方。通俗地说，“指针+整数”用于将指针向后移动“sizeof(指针类型)*整数”个内存单元，而“指针–整数”用于将指针向前移动“sizeof(指针类型)*整数”个内存单元。以 short 型指针 p 为例，short 型数据占据 2 个内存字节，则对 p 的运算如图 8-3 所示。

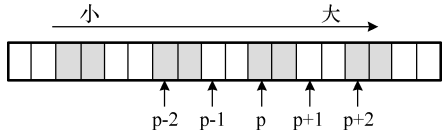


图 8-3　指针与整数相加减示意图

对指针的算术运算使得指针以某数值为单位在内存中前后移动，但编译器并不会检查这种移动的有效性，即目的地址是否可用。如果移动失误，很有可能会修改一些本不该修改的内存单元，给程序带来致命后果。

因此，这种“指针和整数的加减运算”适宜在数组内进行，或者是动态申请的内存。关于动态内存申请的概念将在后续章节中进行介绍。

牢记：一定要让指针处于可控范围内，避免发生乱指一气的现象。

举一个例子来表示指针与整数算术运算的用法，见例 8.5。

【例 8.5】指针变量与整数算术运算 PointerInteger。

文件名：PointerInteger.c

```
01  #include <stdio.h>                  /*使用 printf 要包含的头文件*/
02  #include <conio.h>
03  main()                              /*主函数*/
04  {
05   int sz[9]={1,2,3,4,5,6,7,8,9};/*声明一大小为 9 的 int 型数组 sz，并对其中
                                        元素初始化*/
06  /*声明 int 型的指针变量 p，用数组第 1 个元素地址为其初始化*/
07  int*  p=&sz[0];
08  for(int i=0;i<9;i++)                /*循环*/
09  {
10          printf("%d",*p);            /*间接访问，输出 p 指向单元中的数据*/
11          p++;                        /*等价于 p=p+1*/
12  }
13  getch();                            /*等待，按任意键继续*/
14  }
```

运行结果如下：

```
1   2   3   4   5   6   7   8   9
```

分析：在例 8.5 中，语句“p++;”等价于“p=p+1;”，在声明指针 p 时，用数组 sz 第一个元素的地址为其初始化，之后，每对 p 加 1，p 便指向数组中的下一个元素，通过一个 for 循环，实现了对数组所有元素的输出。

8.4.2 指针-指针

指针变量所支持的另一种运算方式是两个同类型的指针相减，返回值是个有符号的整数，其值可用下列公式计算：

(指针 1 的值–指针 2 的值)/指针所指类型占用的内存字数

指针相减多应用于同一块内存（如数组或一块动态申请的内存）中。如果两个指针所指向的元素没有结构上的关系，指针相减的结果将是不可预测的。打个比方来说，对一条街上的两个门牌号码相减，大致可以判断出中间隔了多少间房子，而在不同街道甚至是不同城市的门牌号码相减，是没有什么实际意义的。

请看例 8.6。

【例 8.6】同类型指针相减 MinusPtrs。

文件名：MinusPtrs.c

```
01  #include  <stdio.h>      /*使用 printf 要包含的头文件*/
02  #include  <conio.h>
03  main()                   /*主函数*/
04  {
05      int sz [5] ={1,2,3,4,5};    /*声明 int 型数组 sz，大小为 5*/
06      int *p1=&sz[1];      /*声明 int 型指针 p1，并用第 2 个元素地址为其赋值*/
```

```
07      int*p2=&sz[4];      /*声明 int 型指针 p2，并用第 5 个元素地址为其赋值*/
08      int  d=p1-p2;       /*同类型指针做差*/
09      printf("p1 的值为:%p\n",p1);      /*输出 p1 的值*/
10      printf("p2 的值为:%p\n",p2);      /*输出 p2 的值*/
11      printf("d 的值为:%d\n",d);        /*输出 d 的值*/
12      getch();                          /*等待，按任意键继续*/
13  }
```

运行结果如下：

```
p1 的值为：0x0012ff60
p2 的值为：0x0012ff6c
d 的值为：-3
```

分析：在例 8.6 中，先声明了一个大小为 5 的 int 型数组 sz，并用 sz[1]和 sz[4]的地址分别为声明的指针变量 p1 和 p2 初始化，p1 的值是 0x0012ff60，而 p2 的值是 0x0012ff6c，从字节数上来看，两者相差的量为 c，即 12 个，但两个指针的距离并不是其值简单做差，还要除以“指针所指类型占用的内存字节”，因此 p2-p1 返回值应当是 12/4=3。为了体现出同类型指针做差返回类型是有符号整数这一特点，例 8.6 使用 p1-p2 为 d 的赋值，所以，返回结果为-3。

规律：在数组中，在类型正确的前提下，若 p1 指向 sz[i]，p2 指向 sz[j]，那么 p1-p2=i-j。

8.4.3　指针的大小比较

对两个毫无相关的指针比较大小是没有意义的，因为指针只代表了“位置”这么一个信息。但是，如果两个指针所指向的元素位于同一个数组（或同一个动态申请的内存）中，指针的大小比较却能反映元素在数组中的先后关系。

举例说明，对例 8.5 进行改写，如例 8.7 所示。

【例 8.7】指针大小比较 PointerComparation。

```
文件名：PointerComparation.c
01  #include<stdio.h>                          /*使用 printf 要包含的头文件*/
02  #include<conio.h>
03  main()                                     /*主函数*/
04  {
05      int  sz[9]={1,2,3,4,5,6,7,8,9};        /*声明一大小为 9 的 int 型数组 sz，
                                                 并对其中元素初始化*/
06      for (int*p=&sz[0];p<=&sz[8];p++)       /*循环*/
07      {
08          printf("%d",*p);                   /*间接访问，输出 p 指向单元中的数据*/
09      }
10      getch();                               /*等待，按任意键继续*/
11}
```

运行结果如下：

```
1   2   3   4   5   6   7   8   9
```

分析：例 8.7 将指针 p 作为 for 循环的循环变量，循环的结束条件为“p<=&sz[8]”。在本例中，比较 p 和数组中各元素的地址是有意义的，因为指向的元素都位于同一个数组中。

实际上，在 for 循环结构执行后，p 指向了 sz[8]后面的那个内存位置，但后面没有利用指针 p

对该块内存进行间接访问操作，因而例 8.7 是安全的，若不小心在循环后对 p 进行间接访问操作，则会引发内存越界访问错误，严重时可能会引起程序崩溃。因此，推荐在 for 循环结束后使用如下语句将指针置空：

```
p=NULL;
```

这样便可有效防止对内存的误操作。

提示：编辑器并不指明指针越界等可能的错误，保证程序安全的责任落在开发人员身上，确保指针指向有意义的内存是最关键的。

8.5 指针表达式与左值

左值是 C 语言中的一类表达式，“左”（left）的原意是指可以放在赋值符号“=”的左边，表示存储在计算机内存的对象。指针表达式应该如何书写才合法，指针表达式能否作为左值，是本节要讨论的内容。

8.5.1 指针与整型

已经提及，在 32 位系统中，无论是何种类型的指针，都占据 4 个内存字节，指针的值是某个内存的地址，这应当是个“整数”。但是，简单地把整数赋给指针也是不允许的，下列代码是错误的：

```
Int* pNum=0x0012FF7C;
```

如果实在有必要对某个内存地址进行访问，可以通过强制类型转换来完成，例如：

```
int* pNum=(int *)0x0012FF7C;
```

8.5.2 指针与左值

指针变量以及指针变量的间接引用都可作为左值，例如：

```
int  num1=0,num2=0;
int* p=&num1;
P=&num2                    /*指针作为左值*/
*p=1;                      /*间接引用作为左值*/
```

指针变量可以作为左值，并不是因为它们是指针，而是因为它们是变量。

8.5.3 指针与 const

const 取自英语单词 constant，是“恒定，不变”的意思，用户可用其修饰变量或函数的参数列表及返回值，限定其不允许改变。使用 const 在一定程度上可以提高程序的健壮性。另外，在观看别人代码的时候，清晰地理解 const 所起的作用，对理解对方程序也有一些帮助。

早期的 C 语言中并没有 const 这个关键字，随着 C 语言的发展，才逐步添加到标准中。使用 const 修饰指针时，通过在不同位置使用 const，可以达到如下三个目的。

（1）禁止对指针赋值。

（2）禁止通过间接引用（*指针）对指针所指的变量赋值。

（3）既禁止对指针赋值，又禁止通过间接引用（*指针）对指针所指的变量赋值。

1．禁止改写指针（常量指针或常指针）

在声明一个指针时，如果在*右边加 const 修饰符，所声明的指针称为常量指针（常指针），编译器不允许程序改写该指针的值。换言之，该指针恒指向某个内存地址，例如：

```
int  x=0;
int* const pInt=&x;
```

上述代码声明了一个指向 int 形变量的指针 pInt，并用 int 型变量 x 的地址为其初始化，在整个执行过程中，pInt 的值无法改变，也就是说，用户无法在后续代码中让 pInt 指向别的内存单元。

注意：无法改写 pInt 并不意味着无法通过间接引用改写 pInt 指向的变量，下述代码是合法的：

```
X=5
*pInt=6
```

声明一个常指针，必须对其进行初始化，因为常指针的值在声明完毕后无法修改，因此，未进行初始化的常指针是没有意义的，编译器将给出错误提示。

2．禁止改写间接引用

在指针声明时，将 const 修饰符放在指针类型符之前，便无法通过间接引用改写指针所指的变量，例如：

```
int x=5
const int*pInt=&x;
```

与常指针不同的是，此处的 pInt 并不被禁写，用户可使 pInt 指向其他的内存单元。但是，通过间接访问（*pInt）改写指针所指的变量是非法的，例如：

```
*pInt=10
```

禁止改写间接引用，并不意味着该内存变量无法改写，通过变量名访问和改写该内存区域是合法的，例如：

```
x=10
```

8.6　动态内存分配

以前的示例程序都是将指针初始化为变量的地址(或用变量的地址来对指针变量赋值)。此外，C 语言函数库中提供了 malloc 和 free 函数，允许动态申请需要的内存，给程序的设计带来了很大的灵活性。

8.6.1　动态分配的好处

先要搞明白一个问题：什么是动态分配，什么是静态分配？举例来说，在声明组数时，必须明确告诉编译器数组的大小，之后编译器就会在内存中为该数组开辟固定大小的内存。类似于组数内存这种分配机制就称为静态分配，很明显，静态分配是由编译器完成的，在程序执行前便已指定。

有些时候，用户并不确定需要多大的内存，为了保险起见，有的用户采用定义一个大数组的方法，开辟的组数大小可能比实际所需大几倍甚至几十倍，这造成内存浪费，很不方便。即使用户确切知道要存放的元素个数，但随着时间规模的变化，数据元素的数目也会变化，个数变少了还好处理，但如果数目增加了，则存储到什么地方去呢？

显而易见，静态分配虽然直观，易理解，但存在明显的缺陷：不是容易浪费内存就是数组的大小不够用。为解决这一问题，C 语言引入了动态分配机制。

动态分配是指用户可以在程序运行期间根据需要申请或释放内存，大小也完全可控。动态分配不像数组内存那样需要预先分配空间，而是由系统根据程序的需要动态分配，大小完全由用户实时指定，当使用完毕后，用户还可释放所申请的动态内存，由系统回收，以备他用，有效地避免了内存浪费。

8.6.2 malloc 与 free 函数

malloc 和 free 是 C 标准库中提供的两个函数，用以动态申请和释放内存，malloc 函数的基本调用格式为：

```
void *malloc(unsigned int size);
```

参数 size 是个无符号整型数，用户由此控制申请内存的大小，执行成功时，系统会为程序开辟一块大小为 size 的内存字节的区域，并将该区域的首地址返回，用户可利用该地址管理并使用该块内存，如果申请失败（比如内存大小不够用），则返回空指针 NULL。

malloc 函数返回类型是 void*，用其返回值对其他类型指针赋值时，必须进行显示转换。

size 仅仅是申请字节的大小，并不需要知道申请的内存块中存储的数据类型，因此申请内存的长度须由程序员通过“长度×sizeof(类型)”的方式给出。举例来说：

```
int* p=(int*)malloc(5*  sizeof(int));
```

系统将开辟一块能存储 5 个 int 数据的内存，并用首地址为刚声明创建的 int 型指针 p 初始化，如果开辟失败，p 将初始化为 NULL。在一般的系统和编译器环境下，上述语句开辟的内存大小为 20 字节。

传递给 free 函数的指针并不是要求一定就是那个接收 malloc 返回值的指针，free 函数关心的是指针的值，即动态内存块的地址，当程序源代码过长时，用户往往会忘记指针的值是从哪里来的，因而多次对同一块动态内存进行释放，引发错误。

8.7 指针与数组

在前面章节中，读者已经领会到了指针的强大功能。实际上，指针所能做的不仅仅是向函数传递变量的地址。本节将讨论一些指针进阶的知识。

8.7.1 数组名指针

数组名是一种常指针（不能修改），其值等于数组占据内存单元的首地址，但其类型取决于数组的维数。

对三维数组 A 而言，有下面的关系成立：

```
A=&A[0];
A+1=&A[1];
…
A[0]=&A[0][0];
A[0]+1=&(A[0][1]);
…
A[0][0]=&A[0][0][0];
```

```
A[0][0]+1=&A[0][0][1];
…
```

更高维的情况以此类推，可见，数组名指针是一种层次关系。

上述关系中的&表示的是一种多级指针对应关系，而非真正取地址，先来看例 8.8 中的代码。

【例 8.8】数组名指针及多级指针的概念 ArrayName。

```
文件名：ArrayName.c
01 #include<stdio.h>                          /*使用printf要包含的头文件*/
02 #include<conio.h>                          /*主函数*/
03 main()
04 {
05     int A[2][3][4]={0};                        /*声明一个 3 维数组*/
06     printf("A is %p\n",A);                     /*数组名指针，3 级指针*/
07     printf("A[0] is %p\n",A[0]);               /*2 级指针*/
08     printf("A[0][0] is %p\n",A[0][0]);         /*1 级指针*/
09     printf("&A[0][0][0] is %p\n",&A[0][0][0]);/*数组首元素的内存地址*/
10     getch();                                   /*等待，按任意键继续*/
11 }
```

运行结果如下：

```
A is 0x0012ff10
A[0] is 0x0012ff10
A[0][0] is 0x0012ff10
&A[0][0][0] is 0x0012ff10
```

分析：就对例 8.8 定义的三维数组 A 而言，数组名 A 实际上是一个 3 级常指针，指向的是一个 3*4 的二维 int 数组结构，同理，A[0]是一个 2 级常指针，指向的是一个大小为 4 的一维数组结构，A[0][0]才是指向 int 数组元素 A[0][0][0]的常指针，但 A，A[0]，A[0][0]的值相同，同为&A[0][0][0]。%P 代表以地址形式输出后续参数。

8.7.2　使用数组名常指针表示数组元素

数组与指针关系密切，数组元素除了可以使用下标来访问，还可用指针形式表示。数组元素可以很方便地用数组名常指针来表示，以三维 int 型数组 A 为例，其中的元素 A[i][j][k]可用下述形式表示。

（1）*（A[i][j]+k）

A[i][j]是 int 型指针，其值为&A[i][j][0],因此，A[i][j][k]可表示为*(A[i][j]+k)。

（2）*(*(A[i]+j)+k)

和第一种形式比较，不难发现 A[i][j]=*A([i]+j),A[i]是 2 级指针，其值为&A[i][0]。

（3）*(*(*(A+i)+j)+k)

将第（2）种形式的 A[i]替换成了*(A+i)，此处 A 是 3 级指针，其值为&A[0]。

此处以三维数组举例，还可以进一步推广到更高维的情况。

8.7.3　指向数组元素的指针变量

使用一个指向数组元素的指针变量可以很方便地访问数组中的元素，不过，在元素定位时需要考虑多维数组在内存中的存储形式。先来看例 8.9 中的代码。

【例 8.9】指向数组元素的普通指针变量 PointertoArray1。

```
文件名：PointertoArray1.c
01 #include<sdtdio.h>              /*使用 printf 要包含的头文件*/
02 #include<conio.h>
03 main()                         /*主函数*/
04 {
05  /*声明一个三维数组 A 并初始化*/
06    int  A[2][3][4]={{{1，2，3，4},
                       {5，6，7，8},
                       {9，10，11，12}},
07                    {{13，14，15，16},
                       {17，18，19，20},
                         {21，22，23，24}}};
08    int* pA=A[0][0];                                  /*为普通指针变量赋值*/
09    for(int i=0;i<2;i++)                              /*第一维*/
10          for(int j=0;j<3;j++)                        /*第二维*/
11          for(int k=0;k<4;k++)                        /*第三维*/
12                printf("%d",*(pA+i*3*4+j*4+k));/*遍历输出*/
13    pA=A[0][0];                                  /*使 pA 重新指向数组开头*/
14    printf("\n");
15   for(int m=0;m<sizeof(A)/sizeof(int);m++)
16   {
17              printf("%d",*pA);                 /*输出 pA 指向元素*/
18              pA++;                             /*指向数组中的下一个元素*/
19   }
20   getch();                                     /*等待，按任意键继续*/
21   }
```

运行结果如下：

```
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21
22  23  24
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21
22  23  24
```

分析：例 8.9 采用两种方法对数组元素进行了遍历，其中第 12 行采用 3 个维数下标来计算指针偏移值，而后一种输出直接对指针 pA 进行递增处理，两者输出的结果一致。

pA 是普通的 int 型指针，除了使用代码 8.9 中的初始化方式外，还可使用如下的初始化方式，下述代码与“int*pA=A[0] [0];”都是等价的：

```
int  *pA=&A[0] [0] [0];                          /*取第一个元素的地址*/
Int  *pA=(int*)A[0];                             /*类型强制转换*/
Int  *pA=(int*)A;                                /*类型强制转换*/
```

和数组名常指针不同，指针变量的值可以修改，使用也更为灵活。

8.7.4 指向数组的指针变量

请体会本小节和上小节标题的不同，仍以三维数组为例，数组名是 3 级常指针，那能否声明一个变量是 3 级常指针呢？先来看例 8.10。

【例 8.10】指向数组的指针变量 PointertoArray2。

文件名：PointertoArray2.c

```
01   #include <stdio.h>              /*使用 printf 要包含的头文件*/
02   #include <conio.h>
03   main()                          /*主函数*/
04   {
05   /*声明一个三维数组 A 并初始化*/
06      int A[2][3][4]={{{1,2,3,4},{5,6,7,8},{9,10,11,12}},
07                  {{13,14,15,16},{17,18,19,20},{21,22,23,24}}};
08      int (*pA)[3][4]=A;                              /*为普通指针变量赋值*/
09          for(int i=0;i<2;i++)                        /*第一维*/
10          for(int j=0;j<3;j++)                        /*第二维*/
11              for(int k=0;k<4;k++)                    /*第三维*/
12                  printf("%d",*(*(*(pA+i)+j)+k));/*遍历输出*/
13       printf("\n");                                  /*换行*/
14       for(int i=0;i<2;i++)                           /*第一维*/
15          for(int j=0;j<3;j++)                        /*第二维*/
16              for(int k=0;k<4;k++)                    /*第三维*/
17                  printf("%d",pA[i][j][k]);           /*第四维*/
18       getch();                                       /*等待，按任意键继续*/
19     }
```

运行结果如下：

```
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21
22  23  24
1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21
22  23  24
```

分析：例 8.10 中采用第 8 行声明了一个指向 int 型 3*4 数组的指针 pA，并用数组名 A 为其初始化，同样的道理，可以采用“int(*pA)[4];”声明一个指向大小为 4 的一维数组的指针。

例 8.10 声明的数组名是指针 pA，其功用和数组名 A 完全相同，既可以用*(*(*(pA+i)+j)+k)形式访问 A[i][j][k]，也可以直接写成 pA[i][j][k]的形式。

8.7.5　指针数组

指针也可以作为数组中的元素，将一个个指针用数组形式组织起来，就构成了指针数组。指针数组的一个重要应用是处理字符串，见例 8.11。

【例 8.11】使用指针数组管理字符串 ArrayofPPointer。

文件名：ArrayofPointer.c

```
01  #include <stdio.h>          /*使用 printf 要包含的头文件*/
02  #include <conio.h>
03  main()                      /*主函数*/
04  {
05  /*声明一个指针数组*/
06  char* pA[7]={"Sunday","Monday","Tuesday","Wednesday","Thursday",
     "Friday","Saturday"};
```

```
07   /*输出提示信息*/
08   printf("今天是一周中的第几天？(周日为 0,周一为 1,...)\n");
09   int index=0;                          /*声明 int 型变量 index，用以标志第几天*/
10   scanf("%d",&index);                   /*读取用户输入*/
11   printf("今天是%s",pA[index]);         /*输出英文单词*/
12   getch();                              /*等待，按任意键继续*/
13  }
```

运行结果如下：

```
今天是一周中的第几天？（周日为 0,周一为 1,...）
4                                  (键盘输入)
今天是 Thursday
```

分析：例 8.11 中声明了一个 char 型指针数组 pA，大小为 7，并用常量字符串的地址为数组元素赋初值，根据用户输入的索引，输出对应的字符串。

形如"Sunday"等字符串称为常量字符串，编辑器将为其在只读存储区分配内存，编辑器将其解释为指向该字符串的指针。

不使用指针数组，改为二级 char 型数组也可以实现同样的功能，但数组维数的大小必须以最长的单词 Wednesday（9 个字符加 1 个末尾空字符=10）为准，即

```
char A[7][10]={"Sunday",
               "Monday",
               "Tuesday",
               "Wednesday",
               "Thursday",
               "Friday",
               "Saturday"};
```

对比不难发现，使用指针数组管理字符串能有效地节省内存。

由于数组这种数据组织方式要求元素类型一致，因此，原则上指针数组中的指针应是同类型的。但是，由于所有的指针均为 4 字节，因此，必要时可通过强制转换来跳过这一约束。

8.8 指针、结构体和结构体数组

结构体变量占据一定内存大小，可声明一个结构体类型的指针，其值为结构体占据内存空间的首地址。而且，一个个的结构体变量可以组织在一起构成结构体数组。本节讨论如何使用指针访问结构体和结构体数组。

8.8.1 两种访问形式

完成结构体的定义后，结构体名可以看成一种新的类型，通过结构体名可以声明指向结构体类型的指针，并可用某个结构体变量的地址为其赋值，举例来说：

```
struct person
{
        char name[20];
        int age;
        char email[50];
```

```
}zangsan={"zang san",24,zs@163.com},  *pzs=&zangsan;
```

则 pzs 是一个指向结构体类型的指针，pzs 的值为结构体变量 zangsan 所占据内存的首地址。使用指针访问结构体成员有以下两种形式：

（1）(*指针).成员

（2）指针->成员

两种访问方式是等价的。

8.8.2　声明创建一个结构体数组

数组占据的是一片连续的内存空间，而且，数组元素的类型是一致的。由此可知，创建一个结构体数组是完全可行的，和 C 语言内置类型（如 int,char 等）数组一样，结构体同样要先“声明”，后“使用”。

数组声明用于通知编译器为该数组开辟特定大小的内存，数组所占据的大小取决于数组声明时指定的元素数目和元素类型。和普通的数组声明一样，结构数组声明的一般格式为：

结构类型名　结构数组名[元素个数]；

仍以上面定义的结构 person 为例，下述语句用于声明一个 person 类型的数组 psz：

```
struct person psz[5];
```

上述语句告诉编译器：psz 是一个数组，其中存储的元素为 person 型，大小为 5。请为这个数组开辟所需要的内存。

8.8.3　结构数组的初始化

和普通数组一样，结构数组既可以在声明的同时完成数组元素的初始化，也可以在数组创建完毕后，对其中的元素进行赋值。

首先来看一下如何在声明的同时对元素进行初始化，例如：

```
struct person psz[3]=
{{"zs",24,"zs@1.com" }{"Ls",25,"ls@2.com"},{"Ww",26,"Ww@3.com"}};
```

下面看例 8.12 中的代码。

【例 8.12】结构数组元素初始化 StructArrayElementlnitial。

```
文件名：StructArrayElementInitial.c
01 #include <stdio.h>          /*使用 printf 要包含的头文件*/
02 #include <conio.h>
03 struct person               /*定义 person 结构*/
04 {
05         char name[20];
06         int age;
07 };
08 main()                      /*主函数*/
09 {
10   /*声明一个三维数组 psz1 并初始化，合法*/
11   struct person psz1[3]={{"zs",24},{"Ls",25},{"Ww",26}};
12   /*声明一个 person 类型的结构变量 p1*/
13   struct person p1={"zs",24};
14   /*试图用变量为数组元素初始化，非法*/
```

```
15  /* struct person psz2[3]={p1,p1,p1};*/
16  /*当所有元素都被初始化时，声明时可省略数组大小*/
17  struct person psz3[ ]={ {"Zs",24},{"Ls",25},{"Ww",26} };
18  for(int i=0;i<3;i++)
19  {
20       printf("%s\n",psz1[i].name);      /*遍历输出*/
21       printf("%s\n",psz3[i].name);      /*遍历输出*/
22   }
23  getch();                               /*等待，按任意键继续*/
24  }
```

运行结果如下：

```
Zs
Zs
Ls
Ls
Ww
Ww
```

分析：由例 8.12 可以看出，如果所有元素都被初始化，编译器可自行判断组数的大小，此时在组数声明时，不用指定数组元素的个数，对应上述第 17 行代码。

例 8.12 中的第 15 行是被注释掉的代码。读者可以试着去掉该句代码的注释，看是否可编译通过。试后可知，编译器报错，C 语言不允许使用变量为数组元素进行初始化。

8.8.4 结构数组的使用

声明创建一个结构数组并对其中的元素初始化后，便可以像使用普通数组那样（“数组名+下标”的形式）使用结构数组，同时，上节中介绍的指针与数组相关内容对结构体指针和结构体数组来说完全适用。合理地使用结构体变量、结构体指针和结构体数组，能有效地解决复杂问题。

8.8.5 指向结构数组的指针

普通数组名可以看成“指向数组元素首地址的常指针”，结构数组名同样可以看成指向结构数组元素首地址的常指针，也可以声明一个结构指针变量，使其指向结构数组元素的首地址。这两种方式都能实现通过指针访问结构数组的元素。看例 8.13 中的代码。

【例 8.13】结构数组名常指针和指向结构数组的指针变量 PointerToStructArray。

文件名：PointerToStructArray.c

```
01  #include <stdio.h>          /*使用 printf 要包含的头文件*/
02  #include<conio.h>
03  struct person               /*定义 person 结构*/
04  {
05     char name[20];
06     int age;
07  };
08  main()                      /*主函数*/
09  {
10       /*声明一维组数 pszl 并初始化，合法*/
```

```
11      struct person pszl[3]={"zs",24,"LS",25,{"Ww",26}};
12      /*声明一 person 类型的结构指针变量 pl，并用数组名为其初始化*/
13      struct person* pl=pszl;
14      for(int i=0;i<3;i++)
15      {
16         printf("%s\n",(pszl+i)->name);   /*遍历输出*/
17         printf("%s\n",(*pl).name);       /*遍历输出*/
18         pl++;
19       }
20       getch();                /等待，按任意键继续*/
21  }
```

运行结果如下：

```
Zs
Zs
Ls
Ls
Ww
Ww
```

分析：例 8.13 中，pszl 是结构数组名，也是指向结构数组元素首地址的常指针。所谓常指针，指的是 pszl 不能被改写，因此，使用其访问数组元素时，采用了(pszl+i->name)的形式，当然，也可以采用(*(pszl+i)).name 的形式访问数据成员 name。pl 是声明的结构指针变量，用 pszl 为其赋值使其指向数组首地址，第 13 行实质上等价于“struct person*pl=&psz1[0];”，在循环体内通过代码第 18 行使该指针不断指向下一个元素，实现了对整个数组的遍历访问。在访问结构数据成员 name 时，示例采用了(*pl).name 的形式。由 8.8.1 节的介绍可知，此处还可写成 pl->name。

8.9　函 数 指 针

8.9.1　函数名指针

就像数组名是指向数组元素首地址的常指针一样，函数名也是指向函数的指针，函数在内存中也有对应的一块存储单元，函数名便是指向该块内存的常指针。换句话说，通过函数名确定要执行的代码块在内存中的位置。

做一个小实验来验证上面的结论，看例 8.14 中的代码。

【例 8.14】输出函数名指针 FunctionName。

文件名：FunctionName.c

```
01  #include <stdio.h>          /*使用 printf 要包含的头文件*/
02  #include <conio.h>
03  main()                      /*主函数*/
04  {
05      void disp(void);        /*函数声明*/
06      printf("%p",disp);      /*函数名输出*/
07      disp();                 /*函数执行*/
08      getch();                /*等待，按任意键继续*/
```

```
09  }
10  void disp()                    /*函数定义*/
11  {
12        printf("\n Hello,c");
13  }
```

运行结果如下：

```
0×004012f5
Hello, c
```

提示：内存分配随编译器和操作系统的不同有所不同，因此在读者的机器上，输出的地址可能与此处给出的结果不同。

分析：例 8.14 中定义了一个非常简单的函数 disp，该函数只输出一句简单的“Hello, c”，输出函数名 disp 可以发现，编译器将其解释为一个内存地址，这个地址是该函数可执行代码在内存中的位置。

8.9.2 指向函数的指针

下面解决另一个问题：有没有指向函数的指针变量？答案是“有”。和函数名常指针不同，函数指针变量的值可以改变。换句话说，只要满足特定的条件，既可以让声明的函数指针变量指向 A 函数，也可以让其指向 B 函数。

函数指针变量的声明格式与普通指针变量的声明格式有所不同，来看下述两个语句：

```
int*  fun( int );
int  (*fun)( int );
```

第一条语句容易理解，fun 是函数名，该函数所带参数是 int 型，返回 int 型指针类型；第二条语句就声明了一个函数指针，即指向函数的指针 fun，该函数返回值是 int，所带的参数也是 int 型。

这里仅仅相差一个括号，意义便大不一样。

函数指针变量声明完毕后，其仅仅是个无所指向的指针，对其进行初始化或赋值是非常重要的，这样，才能用该指针调用其他函数。当然，复制和初始化的前提是，待调用函数和所声明的函数指针变量在返回值和参数上一致。来看例 8.15 中的代码。

【例 8.15】指向函数的指针变量 PointerToFunc。

```
文件名：PointerToFunc.c
01  #include <stdio.h>          /*使用 printf 要包含的头文件*/
02  #include <conio.h>
03  main()                      /*主函数*/
04  {
05   char sz1[ 20 ]="I Love";   /*声明一个 c 风格字符串 sz1*/
06    char sz2[ 20 ]="China";   /*声明一个 c 风格字符串 sz2*/
07   char* (*pFun) (char*,char*);   /*声明函数指针 pFun*/
08   pFun=strcat;               /*函数名常指针为函数指针变量赋值*/
09   (*pFun)  ( sz1, sz2 );     /*间接引用形式调用函数*/
10   printf("%s",sz1);          /*输出*/
11   pFun = &strcpy;            /*对函数名取地址，与 pFun=strcat 完全一致*/
12    pFun (sz1,sz2);           /*直接用指针调用也可*/
13    printf("\n%s",sz1);       /*输出*/
```

```
14    getch();                          /*等待，按任意键继续*/
15  }
```

运行结果如下：

```
I Love China
China
```

分析：例 8.15 调用了两个处理字符串的函数 strcat 和 strcpy，通过表 8-1 回顾一下。

表 8-1　strcpy 函数与 strcat 函数

操作	函数原形
复制 C 风格字符串	Char* strcpy(char*, char*)
连接两个 C 风格字符串	Char* strcat(char*,char*)

代码第 7 行声明了一个函数指针 pFun，声明完毕后，pFun 只是个空指针，并没有指向实际的函数，因此，采用了如下形式为 pFun 赋值：

```
pFun=strcat;
pFun=&strcpy;
```

这样才能根据指针 pFun 调用两个函数。注意，对函数来说，上述两种写法是等价的，读者可自行尝试输出 strcat 和 strcpy 进行检验，这和数组有些类似，以三维数组(int A[2][3][4])来说，A、&A、A[0]、&A[0]、A[0][0]、&A[0][0]的值均相同，等于 A[0][0][0]的地址(&A[0][0][0])。

注意：之所以能使用 strcat 和 strcpy 为 pFun 赋值，是因为这两个函数的返回类型和参数类型与 pFun 的声明格式一致。

再来看一下如何用指针 pFun 调用其指向的函数，下面两种格式也是等价的：

```
(*pFun)(sz1,sz2);
pFun (sz1,sz2);
```

也就是说，既可以像普通指针用法那样，使用间接引用(*pFun)的形式调用函数，也可以直接将函数指针当函数名来调用。就例 8.15 来说，首先使 pFun 指向 strcat，这会将 sz2 连接到 sz1 后，输出 sz1 即会在屏幕上输出连接后的字符串“I Love China”，随后让 pFun 指向 strcpy，将 sz2 复制给 sz1，这将会冲掉 sz1 中原有的内容“I Love China”，所以第二次输出 sz1，换行后在屏幕上输出“China”。

另外，不仅可以采用赋值的方式使函数指针变量指向特定的函数，在函数指针声明的同时也可进行初始化，在例 8.15 中：

```
char* (*pFun)(char*,cahr*);      /*声明函数指针 pFun*/
pFun=strcat;                     /*函数名常指针为函数指针变量赋值*/
```

完全可以写成下述形式：

```
char* (*pFun) (char*,char*) = strcat;
```

或

```
char* (*pFun) (char*,char*) = &strcat;
```

8.9.3　函数指针数组

先复习一下指针数组的概念，当数组元素都是同种类型的指针时，该数组为指针数组，如“int* A[3];”即声明了一个指针数组 A，大小为 3，其中每个元素都是 int 型指针。如果数组元素都是指向同型函数（返回值类型相同，参数类型相同）的指针，则该数组称为函数指针数组。来看一个例子：

```
double(*f[5])();
```

f 是一个数组，有 5 个元素，元素都是函数指针，指向没有参数且返回 double 类型的函数，函数指针数组的使用方式和普通数组完全一致。

【例 8.16】函数指针数组的应用 ArrayOfFuncPtr。

文件名：ArrayOfFuncPtr.c

```
01  #include <stdio.h>                  /*使用 printf 要包含的头文件*/
02  #include <conio.h>
03  main()                              /*主函数*/
04  {
05      char sz1[20]="I Love";          /*声明一个 c 风格字符串 sz1*/
06      char sz2[20]="China";           /*声明一个 c 风格字符串 sz2*/
07       /*声明函数指针数组 fun，并对其中的元素进行初始化*/
08      char*  (*fun[2])(char*,char*)={strcat,strcpy};
09      fun[0](sz1,sz2);                /*函数调用*/
10      printf("%s",sz1);               /*输出*/
11      fun[1](sz1,sz2);                /*函数调用*/
12      printf("\n%s",sz1);             /*输出*/
13      getch();                        /*等待，按任意键继续*/
14  }
```

运行结果如下：

```
I  Love  China
China
```

分析：例 8.16 中，第 8 行声明了一个函数指针数组 fun,有两个元素，每个元素都是指向返回 char*，带两个 char*参数的函数指针，同时，用 C 标准库函数名常指针 strcat 和 strcpy 为两个元素初始化。因此，后面便可以采用下述形式调用 strcat 和 strcpy 函数：

```
fun[0](sz1,sz2);
fun[1](sz1,sz2);
```

8.9.4 指向函数指针的指针

再来看下述语句：

```
double (*f[5])();
```

已经知道，数组名可作为指向数组首元素起始地址的常指针，那么函数指针数组的数组名是什么呢？类推得出，函数指针数组名，对应上面语句中的 f，是指向函数指针的常指针。下述代码声明了一个指向函数指针的指针变量 p，并用 f 为其初始化:

```
double (**p)()=f;
```

本 章 小 结

计算机中的每个内存单元都有一个标识，对 C 语言来说，short、int 等内建类型占据着不止一个内存单元，指针指向的是某个量在内存中的首地址。

通过指针可以间接访问其指向的内存区域，此时要用操作符*，在指针声明时也要用到操作符*，但在两个场合下其作用不同，要注意区分。

声明一个指针时，一定要注意对其初始化，使其指向有意义的区域，未经初始化的指针的值是随机的，对其指向的内存区域进行间接访问，结果往往是不可预测的。如果指针实在无处可指，

可将指针设定为 NULL，用于通知系统该指针不指向任何地方。指针变量支持少量的运算，主要有指针和整数的加减、同类型指针求差、指针关系比较等。

C 语言还提供了动态内存分配机制以方便用户管理内存，在使用结束后，应及时将内存释放，避免内存溢出现象的发生。

在 C 语言中，指针是强大的工具，但正如快刀易伤手的道理一样，如果使用不当，可能会给程序带来灾难性的后果。特别是一些错误用法并不会被编译连接器察觉，到了执行阶段才逐渐体现，如内存溢出、非法内存访问等，因此，掌握指针的用法是 C 语言初学者感觉头疼但却必须吃透的难关。

本章还讨论了数组名常指针，指向数组的指针和指针数组的相关内容，指针不仅仅适用于 C 语言内置的类型，如 int,double 等，对自定义的类型，如结构体和共用体等，同样可以利用指针。换句话说，是其在内存中的位置对其进行访问，结构体变量和共用体变量也可以作为数组的元素，构成结构数组。对结构体来说，访问其内部有两种方式：一种是“结构变量名.xxx”，另一种是“指向结构变量的指针->xxx”。

函数指针是本章讨论的另一个重要内容，数组名被解释为指向数组元素首地址的常量指针，函数名同样可解释为指向函数在内存中对应的可执行代码块首地址的常指针。可以声明指向函数的指针，该指针可以像函数名一样使用来调用其指向的函数。

同时，函数指针可以作为另一个函数的参数，以完成特定的功能，函数指针也能构成函数指针数组，以合理组织程序结构。

习　题　8

一、填空题

1．在 C 程序中，可以通过三种运算方式为指针变量赋地址值，它们是_____,=, malloc。

2．在 C 程序中，只能给指针赋 NULL 值和_____值。

3．若有以下定义和语句：int a[4]={0,1,2,3},*p; p=&a[2];，则*--p 的值是_____。

4．将数组 a 的首地址赋给指针变量 p 的语句是_____。

5．将函数 funl 的入口地址赋给指针变量 p 的语句是_____。

6．执行以下程序段后，s 的值是_____。

```
int a[]={5,3,7,2,1,5,3,10},s=0,k;   for(k=0;k<8;k+=2)     s+=*(a+k);
```

7．设有以下定义的语句:int a[3][2]={10,20,30,40,50,60}，(*p)[2];p=a;，则*(*(p+2)+1)值为_____。

8．已知 a=10，b=15，c=1，d=2，e=0，则表达式!a<e 的值为_____。

9．若有以下定义和语句：int a[5]={1,3,5,7,9},*p;p=&a[2];，则++(*p)的值是_____。

10．执行下列语句后，*(p+1)的值是_____。

```
char  s[3]="ab",*p;
p=s;
```

二、单选题

1．关于指针的概念说法不正确的是（　　）。

A．一个指针变量只能指向同一类型变量

B．一个变量的地址称为该变量的指针

C．只有同一类型变量的地址才能放到指向该类型变量的指针变量之中

D．指针变量可以用整数赋值，不能用浮点数赋值

2．若已定义 x 为 int 类型变量，下列语句中说明指针变量 p 的正确语句是（　　）。

A．int p=&x;　　B．int *p=x;　　C．int *p=&x;　　D．*p=*x;

3．下面选项中正确的赋值语句是（设 char a[5],*p=a;）（　　）。

A．p="abcd";　　B．a="abcd";　　C．*p="abcd";　　D．*a="abcd";

4．若有 int a[][]={{1,2},{3,4}};，则*(a+1),*(*a+1)的含义分别为（　　）。

A．非法,2　　B．&a[1][0],2　　C．&a[0][1],3　　D．a[0][0],4

5．已知 p,p1 为指针变量，a 为数组名，j 为整型变量，下列赋值语句中不正确的是（　　）。

A．p=&j,p=p1;　　B．p=a;　　C．p=&a[j];　　D．p=10;

6．若有定义:char *p1,*p2,*p3,*p4,ch;，则不能正确赋值的程序语句为（　　）。

A．p1=&ch; scanf("%c",p1);　　B．p2=(char *)malloc(1);scanf("%c",p2);

C．*p3=getchar();　　D．p4=&ch;*p4=getchar();

7．两个指针变量不可以（　　）。

A．相加　　B．比较　　C．相减　　D．指向同一地址

8．变量 p 为指针变量，若 p=&a，下列说法不正确的是（　　）。

A．&*p==&a　　B．*&a==a　　C．(*p)++==a++　　D．*(p++)==a++

9．以下 read 函数的调用形式中，参数类型正确的是（　　）。

A．read(int fd,char *buf,int count)　　B．read(int *buf,int fd,int count)

C．read(int fd,int count,char *buf)　　D．read(int count,char *buf,int fd)

10．具有相同类型的指针变量 p 与数组 a，不能进行的操作是（　　）。

A．p=a;　　B．*p=a[0];　　C．p=&a[0];　　D．p=&a;

三、判断题

1．char *p="girl";的含义是定义字符型指针变量 p，p 的值是字符串"girl"。（　　）

2．int i,*p=&i;是正确的 C 说明。（　　）

3．语句 int*pt 中的*pt 是指针变量名。（　　）

4．可以将一个整型变量赋给一个指定变量。（　　）

5．调用函数不能改变实参指针变量的值，但可以改变实参指针变量所指变量的值。（　　）

6．数组的指针是指数组的起始地址，数组元素的指针是指数组元素的地址。（　　）

7．引用数组元素只能用下标法。（　　）

8．设有定义：int(*ptr){10}; 其中的 ptr 是一个指向具有 10 个元素的一维数组的指针。（　　）

9．指针不允许进行乘除运算。（　　）

10．移动指针时，不允许加上或减去一个非整数。（　　）

四、程序填空题

1．有 n 个人围成一圈，顺序排号。从第一个人开始报数（从 1～3 报数），凡报到 3 的人退出圈子。问：最后留下的那位是原来的第几号？

```
#define nmax 50
main()
```

```
{int i,k,m,n,num[nmax],*p;
printf("please input the total of numbers:");
scanf("%d",&n);
p=num;
/***********SPACE***********/
for(i=0;【?】;i++)
/***********SPACE***********/
*(p+i)=【?】;
i=0;
k=0;
m=0;
while(m<n-1)
{/***********SPACE***********/
if(【?】!=0) k++;
if(k==3)
{ *(p+i)=0;
k=0;
m++;}
i++;
if(i==n) i=0;}
/***********SPACE***********/
while(【?】) p++;
printf("%d is left\n",*p);}
```

2．输出两个整数中大的那个数，两个整数由键盘输入。

```
#include "stdio.h"
void main()
{int a,b,*p1,*p2;
/***********SPACE***********/
 p1=【?】malloc(sizeof(int));
 p2=(int*)malloc(sizeof(int));
/***********SPACE***********/
 scanf("%d%d",【?】,p2);
 if(*p2>*p1)*p1=*p2;
 free(p2);
/***********SPACE***********/
 printf("max=%d\n",【?】);}
```

五、程序改错题

说明：

① 每题 3 或 4 个错；改错时不可以增加或删除程序行，也不可以更改程序的结构。

② 表示形式/**********FOUND*********/，错误在接下来的一行内。

1．给定程序 MODI1.C 中函数 fun 的功能是：输入两个双精度数，函数返回它们的平方和的平方根值。例如输入 22.936 和 14.121，输出为 y = 26.934 415。

```
#include <stdio.h>
#include <conio.h>
```

```
#include <math.h>
/**********FOUND**********/
 double fun (double *a, *b)
{ double c;
/**********FOUND**********/
  c = sqr(a*a + b*b);
/**********FOUND**********/
return *c;
}
main()
{ double a, b, y;
clrscr();
printf ("Enter a, b: ");
scanf ("%lf%lf", &a, &b);
y = fun (&a, &b);
printf ("y = %f \n", y);
}
```

2．给定程序 MODI1.C 中函数 fun 的功能是：求两个形参的乘积和商数，并通过形参返回调用程序。例如，输入 61.82 和 12.65，输出为 c = 782.023000，d = 4.886957。

```
#include <stdio.h>
#include <conio.h>
/**********FOUND**********/
 void fun ( double a, b, double *x, double *y )
{
/**********FOUND**********/
 x = a * b;
/**********FOUND**********/
  y = a / b;
}
main()
{ double a, b, c, d;
clrscr();
printf ("Enter a, b:");
scanf ("%lf%lf", &a, &b);
fun (a, b, &c, &d);
printf ("c=%f,d=%f\n", c, d);
getch();
}
```

六、程序设计题（注意：部分源程序已给出。请勿改动主函数 main 和其他函数中的任何内容，仅在函数的花括号中填入所编写的若干语句。）

1．编写一个函数 fun，函数的功能是：输入一个字符串，过滤此串，只保留串中的字母字符，并统计新生成串中包含的字母个数。例如，输入的字符串为 ab234$df4，新生成的串为 abdf。

```
------------------------------------------------
#include <stdio.h>
```

```
#include <conio.h>
#define N 80
main()
{
    char str[N];
    int s;
    clrscr();
    printf("input a string:");gets(str);
    printf("The original string is:"); puts(str);
    s=fun(str);
    printf("The new string is:");puts(str);
    printf("There are %d char in the new string.",s);
    getch();
    yzj();
}
fun(char *ptr)
{
/**********Program**********/

/**********  End  **********/
}
yzj()
{
FILE *IN,*OUT;
char sIN[N];
int iOUT;
IN=fopen("in.dat","r");
if(IN==NULL)
{printf("Please Verify The Current Dir..It May Be Changed");
}
OUT=fopen("out.dat","w");
if(OUT==NULL)
{printf("Please Verify The Current Dir..It May Be Changed");
}
fscanf(IN,"%s",sIN);
iOUT=fun(sIN);
fprintf(OUT,"%d %s\n",iOUT,sIN);

fclose(IN);
fclose(OUT);
}
```

2．编写函数 fun，函数的功能是：从字符串中删除指定的字符，同一字母的大小写按不同字符处理。若程序执行时输入字符串为 turbocandborlandc++，从键盘上输入字符 n，则输出后变为 turbocadborladc++；如果输入的字符在字符串中不存在，则字符串照原样输出。

```
-----------------------------------------------
#include "stdio.h"
```

```
int fun(char s[],int c)
{
/**********Program**********/



/**********  End  **********/
}
main()
{
static char str[]="turbocandborlandc++";
char ch;
clrscr();
printf("原始字符串:%s\n", str);
printf("输入一个字符:");
scanf("%c",&ch);
fun(str,ch);
printf("str[]=%s\n",str);
getch();
yzj();
}
yzj()
{
FILE *IN,*OUT;
char i[200];
char o[200];
IN=fopen("in.dat","r");
if(IN==NULL)
{printf("Read File Error");
}
OUT=fopen("out.dat","w");
if(OUT==NULL)
{printf("Write File Error");
}
fscanf(IN,"%s",i);
fun(i,'n');
fprintf(OUT,"%s",i);

fclose(IN);
fclose(OUT);
}
```

第 9 章　结构体、共用体和枚举

设计程序最重要的一个步骤就是选择一个表示数据的好方法。前面已经介绍了基本类型、指针类型和数组等表示数据的方法。在多数情况下，使用简单的变量甚至数组都是不够的。在实际应用中，有许多不是同一种类型的数据要集合到一起的例子需要处理。

本章的重点是掌握结构体和共用体类型的说明、结构体和共用体变量的定义及初始化；掌握结构体与共用体变量成员的引用，了解枚举类型变量的定义，了解 typedef 的作用。本章的难点是理解链表的基本概念，掌握链表的基本操作。

9.1　结构体类型

C 语言的结构体类型可以灵活地表示不同类型的多种数据，这种表示数据的方法进一步增强了数据的表示能力。例如，图书馆要打印出图书的详细信息，包括书名、作者、出版商、版权日期、页数、册数及价格等。其中的一些项目可以用字符串数组存储，另外的一些则需要用 int 数组或 float 数组存储。如果使用多个不同的数组来保存所有的信息将是比较复杂的。一个好的解决方法是定义一种数据形式，其中既可包括字符串又可包括数字，还能够分别保存这些信息。结构体类型就满足了这种需要。

9.1.1　建立结构体声明

结构体类型（struct）是由若干成员组成的。每一个成员可以是一个基本数据类型，也可以是其他构造类型（结构体类型是构造类型中的一种，其他构造类型将在后面介绍）。结构体是一种数据类型，在说明和使用之前必须先定义它。结构体类型定义的形式为：

```
struct 结构体类型名
{  类型 1 成员 1;
   类型 2 成员 2;
   …
   类型 n 成员 n;
};
```

假设每本书的信息只包括书名、作者和当前的市场价格。使用结构体定义这种数据类型代码如下：

```
struct book
{  char title[40];
   char author[30];
   float value;
};
```

定义结构体类型首先使用关键字 struct。struct 是结构体类型的统一说明和引用，表示接下来是一个结构体类型。后面是一个可选的标记（比如：book），在后面声明这种结构类型变量时，会使用该标记，用来引用该结构。在结构体类型定义中，用一对花括号来包含结构体成员列表。每

个成员变量都用它自己的声明来描述，用分号来结束描述。结束花括号后的分号表示结构体设计定义的结束。

9.1.2 结构体变量的定义

结构有两个意思：一是进行结构体定义，确定结构体包括哪些数据，但是进行结构体定义并不为数据分配空间；二是创建结构体变量。结构体变量定义形式如下：

```
struct 结构体类型名 结构体变量名;
```

例如：

```
struct book library;
```

上面语句定义了结构体类型变量 library。结构体 book 的定义是创建了一个名为 struct book 的新类型，struct book 所起的作用与基本类型 int 或 float 在普通声明变量中的作用一样。看到这条指令，编译器创建一个变量 library，为该变量分配空间。结构体类型变量 library 包括一个名字为 title、长度为 40 的 char 型数组，一个具有 30 个元素、名字为 author 的 char 数组和一个 float 变量 value。

结构体变量所占的存储空间为各个成员变量所占存储空间的和，并且各个成员变量是按类型定义的顺序分配地址空间的。在 TC 环境下，变量 library 所占存储空间为 74 字节。

定义结构体变量时，可以同时定义两个或多个变量，也可以定义一个指向该结构的指针。例如：

```
struct book library, *ptbook;
```

结构体指针 ptbook 是可以指向变量 library 或任何其他结构体 book 的变量。

声明结构的过程和定义结构变量的过程可以被合并成一步。例如：

```
struct book
{  char title [40];
   char author[30];
   float value;
} library, *ptbook;                    /*在定义之后跟变量名*/
```

这段代码在定义结构体类型 book 的同时定义了两个 book 类型的变量 library，以及一个可以指向 book 结构的指针 ptbook。

另外，将声明和变量定义合并在一起，也可以不需要使用标记。

```
struct                          /*不使用类型标记*/
{  char title [40];
   char author[30];
   float value;
} library, *ptbook;
```

但这种方法只能在此处定义结构体变量。因为没有结构体类型名称，这种结构体类型是无法重复使用的。如果需要多次使用一个结构体类型，就需要使用带有标记的形式。

结构体变量也可以在定义时进行初始化。结构体变量初始化与数组初始化格式相似，例如：

```
struct book library = { 'Adventure of Sherlock Holmes', 'Conan Doyle', 19.5};
```

从上面的结构体变量初始化代码中可以看出，结构体变量的初始化需要使用一个用花括号括

起来的并用逗号分隔的初始化项目列表。每个初始化项目必须与要初始化的结构成员类型相匹配，也可以把每个成员的初始化项目写在单独的一行中，例如：

```
struct book library = {
'Adventure of Sherlock Holmes',
'Conan Library',
19.5};
```

9.1.3　结构体变量的引用

结构像一个超级数组。在这个超级数组内，其成员变量类型可以不同。在数组中可以使用下标访问数组的各个元素。结构体变量成员的访问也可以使用结构成员运算符点（.）。结构体成员变量的引用方式为：

结构体类型变量名.成员变量名

例如，library.value 就是指 library 的 value 部分，可以像使用任何其他 float 类型普通变量那样使用 library.value。例如，scanf("%f", &library.value)。其中，点运算符“.”拥有比地址运算符&更高的优先级。因此，这个表达式相当于&(library.value)。

【例 9.1】结构体成员引用举例。

```
#include <stdio.h>
struct book
{  char title[40];
   char author[30];
   float value;
} library;                          /*在定义之后跟变量名*/
int main(void)
{  struct book library;             /*把 library 声明为 book 类型变量*/
   printf ("Please enter the book title:\n");
   gets(library.title);             /*访问 tit1e 部分*/
   printf("Now enter the author:\n");
   gets(library.author);
   printf("Now enter the value:\n");
   scanf ("%f",&library.value);
   printf ("%s \"%s\" ($%.2f) \n", library.author, library.title, library.value);
   return 0;
}
```

运行结果如图 9-1 所示。

```
Please enter the book title:
Adventure of Sherlock Holmes
Now enter the author:
Conan Doyle
Now enter the value:
14.95
Conan Doyle "Adventure of Sherlock Holmes" ($14.95)
```

图 9-1　例 9.1 运行结果

9.2 结构体数组

如果需要处理两本图书，则需用两个结构变量来描述。如果需要处理多本图书，则需用多个结构变量来描述。和普通数组一样，可以使用结构体数组。结构体数组中的每个分量就相当于一个结构体变量，用法与普通结构体变量相同。

声明结构体数组和声明数组的格式一样，如下所示：

```
struct 结构体类型名 数组名[数组长度];
```

例如：

```
struct book library[200];
```

这条语句声明的结构体数组 library 是一个具有 200 个元素的结构体数组，其数组元素依次为 library[0]，library[1]，…，library[199]。注意，下标标号依然从 0 开始。

每个 library[i]元素都是结构体 book 类型，相当于普通的结构体变量。引用结构体数组每个元素的成员的方法是，先在结构名后加一个点运算，然后是成员名。例如，library[0].value 表示第 1 个数组元素的 value 成员。

如果想访问 title 成员数组中的一个字符，则应表示为 library[4].title[3]，表示第 5 个数组元素的 title 成员中第 4 个字符。

【例 9.2】将学生成绩表按总成绩降序排列。

```
#include <stdio.h>
#define N 4
struct student
{  char num[7],name[7];
   int com,eng,total;};
   void sorttotal(struct student s[], int n)
{  int i,j,k;
   struct student t;
   for(i=0;i<n-1;i++)
        {  for(k=i,j=i+1;j<n;j++)
                if(s[k].total<s[j].total)  k=j;
           if(i!=k)
                { t=s[k]; s[k]=s[i]; s[i]=t;}
        }
}
int main(void)
{  int i;
   struct student stu[N]={ {"202149","张娜", 83,77,0},
                           {"203120","李海峰",89,72,0},
                           {"201034","王猛",76,68,0},
                           {"200537","韩世吉",82,74,0}};
   for(i=0;i<N;i++)
        stu[i].total=stu[i].com+stu[i].eng;
   sorttotal(stu,N);
```

```
    printf("学号\t 姓名\t 计算机成绩\t 英语成绩\t 总成绩\n");
    for(i=0;i<N;i++)
        printf("%s\t%s\t%d\t\t%d\t\t%d\n",
           stu[i].num,stu[i].name,stu[i].com,stu[i].eng,stu[i].total);
    return 0;
}
```

运行结果如图 9-2 所示。

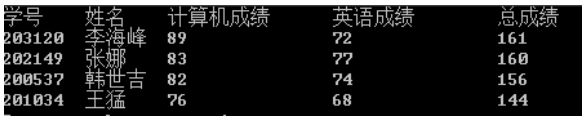


图 9-2　例 9.2 运行结果

9.3　结构体指针

在某些情况下，使用指针更容易操作，表示数据更简单。前面介绍了各种指针和指针变量，有指向普通变量的指针，有指向数组的指针，也有指向函数的指针。这一节将介绍指向结构体类型的指针。

9.3.1　结构体变量的指针

如前所述，可以在定义结构体类型时，声明结构体变量和结构体指针。当然，也可以单独声明结构体指针，形式如下：

```
struct 结构体类型名 *结构体指针变量名;
```

例如：

```
struct guy *him;
```

声明结构体指针首先是关键字 struct，其次是结构体类型标记，然后是指针标记*，紧跟着是指针名。这种声明形式跟其他指针声明一样。这个结构意味着定义了可以指向任何现有的结构体 guy 类型变量的指针 him。

结构体指针和其他指针一样，可以将一个结构体变量的地址赋值给它。例如：

```
him=&barney;
```

其中，barney 为一个 guy 类型的变量。注意，与数组不同的是，结构体变量名字不是该结构的地址，必须使用取地址运算符&。

使用结构体指针访问其指向结构变量成员的方法有以下两种。

第一种方法，也是最常用的方法，是使用指针运算符->。引用形式如下：

```
结构体指针变量名->成员名
```

第二种方法，是使用运算符*。引用形式如下：

```
(*结构体指针变量名).成员名
```

例如，him->income 与(*him).income 是完全等价的。由于点运算符比星号运算符的优先级高，所以在第二种方法中不能省略圆括号。

【例 9.3】结构体指针的使用。

```
#include <stdio.h>
#include <string.h>
struct student
{  int num;
   char name[10];
   char subject[20];};
int main(void)
{  struct student student1, *p;
   student1.num=13;
   p =&student1;
   strcpy((*p).name, "米粒");
   strcpy(p->subject, "音乐");
   printf("学号:%d\n 姓名:%s\n 专业:音乐\n", (*p).num, student1.name,
                                        p->subject);
   return 0;
}
```

```
学号: 13
姓名: 米粒
专业: 音乐
```

运行结果如图 9-3 所示。

图 9-3　例 9.3 运行结果

9.3.2　结构体数组的指针

前面已经介绍过，可以使用指向数组或数组元素的指针和指针变量。同样，对结构体数组及其元素也可以用指针或指针变量来指向。

【例 9.4】结构体数组指针举例。

```
#include <stdio.h>
struct student
{  int num;
   char name[20];
   char sex;
   int age;
};
struct student stu[3]={
{15101,"Li Lin",'M',18},{15102,"Wan Fen",'M',19},{15103,"Liu Min",'F',20}};
int main(void)
{  struct student *p;
   printf(" No.\tName\tSex\tAge\n");
   for(p=stu;p<stu+3;p++)
        printf("%d\t%s\t%c\t%d\n",p->num,p->name,p->sex,p->age);
   return 0;
}
```

```
 No.    Name    Sex     Age
10101   Li Lin  M       18
10102   Wan Fen M       19
10104   Liu Min F       20
```

图 9-4　例 9.4 运行结果

运行结果如图 9-4 所示。

在例 9.4 中，指针 p 是指向结构体数组的指针变量，所以 p++是按数组元素的长度移动的，即它每次移动都是移动到数组中各个结构体分量的首地址。

9.3.3　向函数传递结构信息

与普通变量一样，结构体指针和变量也可以作为函数的参数进行值的传递，略有不同的是，结构体变量的成员也可以作为函数的参数。

具有单个值的数据类型的结构成员可以作为函数参数传递给一个接收这个类型的函数。

【例 9.5】银行客户账户余额计算。

```
#include <stdio.h>
#define FUNDLEN 50
struct funds
{  double bankfund;
   double savefund;
};
double sum(double, double);
int main(void)
{  struct funds stan={3024.72, 9237.11};
   printf("Stan has a total of ￥%.2f.\n",sum(stan.bankfund,stan.savefund));
   return 0;
}
double sum (double x,double y)
{  return (x+y);
}
```

运行结果如下：

```
Stan has a total of ￥12261.83.
```

在例 9.5 中，如果结构体 struct funds 类型定义不变，使用结构体指针作为函数参数，也可以计算出账户余额。

```
double sum(struct funds *);
int main(void)
{  struct funds stan={3024.72, 9237.11};
   printf("stan has a total of ￥%.2f.\n", sum(&stan));
   return 0;
}
double sum (struct funds *money)
{  return (money->bankfund+money->savefund);
}
```

结构体变量也作为函数参数。如果以结构体变量作为函数参数，计算账户余额的代码如下。

```
double sum(struct funds money);
int main(void)
{  struct funds stan={3024.72, 9237.11};
   printf("stan has a total of ￥%.2f.\n", sum(stan));
   return 0;
}
double sum (struct funds money)
{  return (money.bankfund+money.savefund);
}
```

9.4 链表的基本知识

链表是可以根据需要动态地进行存储空间的分配和回收的重要的数据存储结构。它可以存储大量的数据，数据元素之间的关系是通过链接的形式来体现的。每个数据元素以结点的形式存在，结点分为数据域和指针域两部分。数据域根据定义形式可以由一个或多个数据组成，指针域存储与该结点链接的下一个结点的起始地址。根据指针域中指针的个数，链表可以分为由一个指针构成的单（向）链表，由两个指针构成的双（向）链表，等等。图 9-5 是一个最简单的单链表。

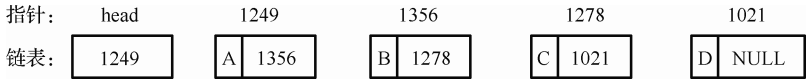


图 9-5 最简单的单链表

链表中有一个称为“头指针”的变量，在图 9-5 中以 head 表示。头指针用来存放链表第一个结点的首地址，也称它指向一个链表。在对链表进行操作时，头指针是必需的，而且不能丢失，否则无法确定整个链表。从图 9-5 可以看出，头指针指向第一个结点，第一个结点又指向第二个结点，…，直到最后一个称为“表尾”的结点。表尾指针域中存放一个空指针，表明它不再指向任何元素，整个链表到此为止。

整个链表就是通过指针顺序链接的，常用带箭头的短线（→）来明确表示这种链接关系，如图 9-6 所示。

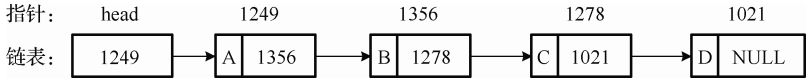


图 9-6 链表关系的表示方法

9.4.1 动态分配和释放空间的函数

C 语言系统提供了几个用于分配和回收存储空间的函数，在链表结点的建立和删除时是必需的。在使用这些函数时必须包含头文件"malloc.h"。

malloc()函数是分配存储空间的函数，其函数原型为：

```
void *malloc(unsigned int size);
```

malloc()函数的作用是，在内存中动态地获取一个大小为 size 个字节的连续的存储空间。该函数将返回一个 void 类型的指针，如果分配成功，则该指针指向已分配空间的起始地址，否则，该指针为空。

calloc()函数是连续空间分配函数，其函数原型为：

```
void *calloc(unsigned n,unsigned size);
```

calloc()函数作用是，在内存中动态获取 n 个大小为 size 个字节的连续的存储空间。该函数将返回一个 void 类型的指针，如果分配成功，则该指针指向已分配空间的起始地址，否则，该指针为空。

realloc()函数是空间再分配函数，其函数原型为：

```
void *realloc(void *addr,unsigned size);
```

其中，addr 是原空间起始地址的指针。realloc()函数的作用是，重新分配一个大小为 size 个字节的连续空间，并将重新分配的空间的起始地址返回。同时，realloc()函数还会将原空间的数据顺序复制到新分配的空间上，并释放原空间。如果分配不成功，则返回一个空指针。

free()函数是空间释放函数，其函数原型为：

```
void free(void *addr);
```

free()函数是释放由 addr 指针所指向的空间，使这段空间又可以被其他变量所用。不用的空间一定要及时地回收，以免浪费宝贵的内存空间。

9.4.2　建立和输出链表

所谓动态建立链表是指在程序执行过程中从无到有地建立链表，将一个个新生成的结点顺次链接入已建立起来的链表中，上一个结点的指针域存放下一个结点的起始地址，并给各个结点数据域赋值。所谓输出链表是将链表上各个结点的数据域中的数据值依次输出，直到链表结尾。

【例 9.6】建立和输出一个学生成绩链表。

```
#include "stdio.h"
#include "malloc.h"
typedef struct student      /*定义链表结点数据类型 ST 和指针类型*STU*/
{  char name[20];
   int score;
   struct student *next;    /*结点指针域*/
}ST,*STU;
STU crelink(int n)          /*建立由 n 个结点构成的单链表，返回结点指针类型*/
{  int i;
   STU p,q,head;
   if(n<=0)     return NULL;         /*参数不合理，返回空指针*/
   head=(STU)malloc(sizeof(ST));     /*生成第一个结点*/
   printf("Input datas:\n");
   scanf("%s %d",head->name,&head->score); /*两数据间用空格间隔*/
   p=head;                           /*p 作为连接下一个结点 q 的指针*/
   for(i=1;i<n;i++)
     {  q=(STU)malloc(sizeof(ST));
        scanf("%s %d",q->name,&q->score);
        p->next=q;            /*连接 q 结点*/
        p=q;                  /*p 跳到 q 上，再准备连接下一个结点 q*/
     }
     p->next=NULL;            /*置尾结点指针域为空指针*/
     return head;             /*将已建立起来的单链表头指针返回*/
     }
list(STU head)                          /*链表的输出*/
{  STU p=head;                          /*依次输出各结点的值*/
   while(p!=NULL)
   {  printf("%s\t%d\n",p->name,p->score);
      p=p->next;                        /*p 指针顺序后移一个结点*/
   }
   return 0;
```

```
}
int main(void)
{  STU h;
   int n;
   printf("Please input number of node:");
   scanf("%d",&n);
   h=crelink(n);              /*调用建立单链表的函数*/
   list(h);                   /*调用输出链表的函数*/
}
```

运行结果如图 9-7 所示。

```
Please input number of node:4
Input datas:
A 10
B 20
C 30
D 40
A       10
B       20
C       30
D       40
```

图 9-7　例 9.6 运行结果

这种建立链表的方法是将新生成的结点从表尾接入，所以也称其为尾接法。对应地，还有将生成的结点插入到链表中第一个结点之前的方法，称为首插法。

9.4.3　链表的基本操作

链表的基本操作是建立、遍历、插入、删除和查找。

1．建立链表的基本操作步骤

（1）生成新结点，q=(STU)malloc(sizeof(ST));。
（2）连接新结点，p->next=q;。
（3）指针跳到新结点上，p=q;。
（4）重复步骤（1）～（3），直到所有结点都建立完毕，转步骤（5）。
（5）安排尾结点的空指针域。

2．遍历链表的基本操作步骤

（1）安排一个指针指向头结点，p=head;。
（2）若 p==NULL，转步骤（5）；否则转步骤（3）。
（3）输出结点值，：printf("%s\t%f\n",p->name,p->score);。
（4）将指针移到下一个结点上，p=p->next，转步骤（2）。
（5）结束。

3．插入结点的基本操作步骤（假定在 p 结点后插入新结点 s）

（1）生成新结点。

```
s=(STU)malloc(sizeof(ST));
scanf("%s %d",s->name,&s->score);
```

（2）插入新结点。

```
s->next=p->next;
p->next=s;
```

【例 9.7】编写一个函数，在例 9.6 中已建立链表的前面插入一个新结点。

```
STU insnode1(STU head)
{  STU s;
   s=(STU)malloc(sizeof(ST));
```

```
        printf("Input new node datas:");
        scanf("%s %d",s->name,&s->score);
        s->next=head;
        head=s;
        return head;
    }
```

将该函数插入到例 9.6 程序的主函数 main()之前，并在主函数中最后加入语句：

```
    h=insnode1(h);
    list(h);
```

运行结果如图 9-8 所示。

```
Please input number of node:3
Input datas:
a 10
b 20
c 30
Input new node datas:W 99
W       99
a       10
b       20
c       30
```

图 9-8　例 9.7 运行结果

4．删除结点的基本操作步骤（假定删除 p 结点的后继结点）

（1）指定欲删除的结点，s=p->next;。

（2）摘掉欲删除的结点，p->next=s->next;。

（3）回收已摘掉的结点，free(s);。

【例 9.8】删除链表中的表首结点函数。

```
STU delnode1(STU head)
{  STU s;
   if(head!=NULL)
   {  printf("After deleted the first node:\n");
      s=head;
      head=s->next;
      free(s);
      }
   return head;
}
```

将该函数放到例 9.7 程序的主函数之前，并在主函数中加上调用语句：

```
    h=delnode1(h);
    list(h);
```

运行结果如图 9-9 所示。

```
Please input number of node:4
Input datas:
a 10
b 20
c 30
d 40
a       10
b       20
c       30
d       40
After deleted the first node:
b       20
c       30
d       40
```

图 9-9　例 9.8 运行结果

链表是存放数据的重要工具，它可以根据需要获取空间，又可以及时回收没用的数据结点，可以存放大量的数据。数组和链表都是用于存放数据元素的结构，它们各有各的优缺点，在具体运用时一定要根据需要选取合适的数据结构。

在存储结构上，数组是一种顺序存储结构，即逻辑上相邻的数据元素在存储地址上也是相邻的；而链表是一种链式存储结构，即逻辑上相邻的数据元素在存储地址上不一定是相邻的。在存取数据上，数组是一种随机存取方式；而链表是一种顺序存取方式。因为链表只能沿着头指针方向顺序往下找，数组则只需通过下标就可以方便地提取相应的数据。在数据处理上，数组非常适合于查找、更新和排序，而链表

则适用于插入和删除操作。因为在链表上进行插入和删除操作时不需要移动大量的数据元素，数组则不然，插入和删除操作平均要移动一半的数据元素。在空间上，数组是事先已限定了固定的空间，不易扩充，而链表则根据需要随时可以获取所需空间。

9.5 共用体类型

共用体（union）类型是一个能在同一个存储空间里（但不同时）存储不同类型数据的数据类型。需要什么类型的数据，这里就存放什么类型的数据。这些数据的起始地址是相同的，数据之间相互覆盖，只有最后一次存入的数据才是有效的。

共用体类型定义的一般形式为：

```
union 共用体类型名
{  类型 1 成员 1;
   类型 2 成员 2;
   类型 3 成员 3;
   …
   类型 n 成员 n;
};
```

例如：

```
union data
{  int i;
   char ch;
   float f;
};
```

从上面共用体类型定义形式上看，它同结构体类型极为相似。不同的是，它说明的几个成员不像结构体那样顺序存储，而是叠放在同一个地址开始的空间上。

共用体类型的长度为最大成员所占空间的长度。例如，共用体类型 union data 的长度为 4 个字节，也就是 float 类型所占的存储空间的长度为 4 个字节。

定义共用体变量的第一种方法是，先定义共用体类型，再定义共用体类型变量，例如：

```
union data a,b,c;
```

也可以在定义共用体类型的同时定义共用体类型变量，例如：

```
union data
{  int i;
   char ch;
   float f;
} a,b,c;
```

以上两种方法都定义了三个共用体类型变量 a、b 和 c。

如果不重复使用已定义的共用体类型，可以在定义无名共用体类型的同时定义共用体类型变量。例如：

```
union
{  int i;
   char ch;
```

```
    float f;
}a,b,c;
```

在以上的三种定义中，编译器将为每个 data 类型变量 a,b,c 分配 4 个字节。

使用共用体类型的数组，可以创建相同大小单元的数组，每个单元都能存储多种类型的数据。例如：

```
union data a[3];
```

这个声明创建了一个 data 数组，含有 3 个元素，每个元素大小为 4 个字节。

与结构体类型一样，也可以定义指向共用体的指针。例如：

```
union data *pointer;
```

这个声明创建了一个指针，可以存放一个共用体 data 类型的共用体变量的地址。

共用体类型初始化的规则与结构体类型初始化不同。共用体初始化有三种方法：一是用一个共用体初始化同类型的另一个共用体，二是初始化共用体的第一个成员，三是初始化一个指定的共用体成员。

```
union data a,b,c;
a.ch='R';
union data a=d;                    //用一个共用体初始化为另一个共用体
union data a={88};                 //初始化共用体的第一个成员
union data a={.f=118.2};           //初始化指定的共用体成员
```

在引用共用体变量成员时，可以使用点运算符.和共用体名称一起来指定共用体的成员，例如：

```
a.i=23;                            //把 23 存储在 a 中，使用 2 个字节
a.ch='a';                          //清除 23，存储'a'，使用 1 个字节
a.f=3.4;                           //清除'a'，存储 3.4，使用 4 个字节
```

在给共用体成员赋值时，在同一时间只能存储一个值。即使有足够的空间，也不能同时保存一个 int 类型和一个 char 类型的值。

在共用体变量中，引用变量成员也可以使用指针->运算符。例如：

```
pointer=&a;
x= pointer->f;                   //相当于 x=pointer.f;
```

【例 9.9】设有一个教师与学生通用的表格，教师数据有姓名、年龄、职业、教研室这 4 项，学生有姓名、年龄、职业、班级这 4 项。编程输入人员数据，再以表格形式输出。

```
#include <stdio.h>
#include <string.h>
int main(void)
{  struct
   {  char name[15];
      int age;
      char job[20];
      union
      {   char major[20];
          char department[20];
      }depa;
```

```
    }body[2];
    int n,i;
    for(i=0;i<2;i++)
    {   printf("input name,age,job:\n");
        scanf("%s%d%s",&body[i].name,&body[i].age,&body[i].job);
        if(strcmp(body[i].job,"student")==0)
        {  printf("input %s's major:",body[i].name);
           scanf("%s",body[i].depa.major);
        }
        else
        {  printf("input %s's department:",body[i].name);
           scanf("%s",body[i].depa.department);
        }
    }
    printf("\n");
    printf("name\t\tage\tjob\t\tmajor/department\n");
    for(i=0;i<2;i++)
    {  if(strcmp(body[i].job,"student")==0)
         printf("%s\t\t%d\t%s\t\t%s\n",body[i].name,body[i].age,body[i].job,
                                                    body[i].depa.major);
        else
         printf("%s\t\t%d\t%s\t\t%s\n",body[i].name,body[i].age,body[i].
                                          job,body[i].depa.department);
    }
    return 0;
}
```

运行结果如图 9-10 所示。

```
input name,age,job:
liyuan 21 student
input liyuan's major:mathematics
input name,age,job:
chenxu 28 teacher
input chenxu's department:computor

name            age     job             major/department
liyuan          21      student         mathematics
chenxu          28      teacher         computor
```

图 9-10　例 9.9 运行结果

9.6　枚 举 类 型

枚举类型（enumerated type）是指通过使用关键字 enum，创建一个新的数据类型，并可以指定它具有的值。定义枚举类型的格式为：

```
enum 枚举类型名
{  枚举常量 1 [=序号 1],
   枚举常量 2 [=序号 2],
   枚举常量 3 [=序号],
```

```
        …
    枚举常量 n [=序号 n]
};
```

例如：

```
enum colors
{  red,
   orange,
   yellow
};
```

为了看上去更方便，往往横着写成这样：

```
enum colors{red, blue, green};
```

在上面声明中，color 为标记名，enum color 将作为一个类型名使用。花括号中的枚举常量列出了枚举类型 color 的所有可能的值，枚举常量也称为枚举元素。各个枚举常量之间要由逗号间隔，而不是分号，最后一个枚举元素的后面无逗号。枚举常量要符合标识符的起名规则。

枚举类型变量的定义格式为：

```
enum 枚举类型名 枚举变量名;
```

例如：

```
enum colors col1;
```

在这个声明中，col1 是枚举类型 color 的一个变量。变量 col1 的可能值是 red、blue、green 等。

在枚举类型中，每个枚举变量对应一个整数。默认时，枚举列表中的常量被指定为整数值 0、1、2 等。在上面的枚举类型 colors 定义中，red 对应为整数 0，blue 对应整数 1，依此类推。也可以设定枚举常量具有的整数值。例如：

```
enum colors{ red=1, blue=2, green=3};
```

如果只对一个常量赋值，而没有对后面的常量赋值，那么这些后面的常量会被赋予后续的值。例如：

```
enum colors{red=100, blue, green=200};
```

则枚举常量 orange 的值为 101。

【例 9.10】请定义枚举类型 score，用枚举元素代表成绩的等级，90 分以上为优（excellent），80～89 分之间为良（good），60～79 分之间为中（general），60 分以下为差（fail）。通过键盘输入一个学生的成绩，然后输出该生成绩的等级。

```
#include<stdio.h>
int main(void)
{  float a;
   enum score{fail,general=6,good=8,excellent}b;
   printf("请输入该学生的成绩：");
   scanf("%f",&a);
   if(a>=60)
        if(a>=60&&a<79)
            b=general;
        else
```

```
            if(a>=80&&a<89)
                b=good;
            else
                b=excellent;
    else
         b=fail;
    switch(b)
    {  case excellent:printf("excellent");break;
       case good:printf("good");break;
       case general:printf("general");break;
       case fail:printf("fail");break;
    }
    printf("\n");
    return 0;
}
```

```
请输入该学生的成绩：87
good
```

图 9-11　例 9.10 运行结果

运行结果如图 9-11 所示。

【例 9.11】定义一个描述三种颜色的枚举类型{red，blue，green}，输出者三种颜色的全部排列结果。

```
#include <stdio.h>
enum colors{ red,blue,green};
void show(colors color)
{
   switch(color)
   {
      case red: printf("%-7s","red"); break;
      case blue: printf("%-7s","blue"); break;
      case green: printf("%-7s","green"); break;
   }
}
int main(void)
{
   colors col1,col2,col3;
   int n=0;
   for(col1=red;col1<=green;col1=colors(col1+1))
      for(col2=red;col2<=green;col2=colors(col2+1))
         for(col3=red;col3<=green;col3=colors(col3+1))
            {   printf("%-4d",++n);
                show(col1);
                show(col2);
                show(col3);
                printf("\n");
            }
   return 0;
}
```

运行结果如图 9-12 所示。

图 9-12　例 9.11 运行结果

引入枚举常量，可以使数据的含义更加清晰易懂。比如上面的问题，用整数 1、2、3 来表示红、蓝、绿来解决问题也是可以的，但不如使用枚举标识符更直观一些。

9.7　typedef 简介

C 语言提供了一个自定义类型的语句——typedef，它能够为某一类型创建新的、方便的、可识别的名称。typedef 和 define 相似，但 typedef 给出的符号名称仅限于对数据类型，而不是对值。另外，typedef 由编译器执行，而不是由预处理器执行。使用 typedef 定义新类型名字的格式为：

```
typedef 原类型名 新类型名;
```

例如：

```
typedef int INTEGER;
typedef float REAL;
typedef unsigned char;
```

这里我们分别给 int 类型、float 类型、unsigned char 类型创建了新的类型名字 INTEGER、REAL、BYTE。在这些定义的后面程序中，定义变量时，int 可以用 INTEGER 来替代，float 类型可以用 REAL 来替代，unsigned char 可以用 BYTE 来替代。例如：

```
INTEGER a;
REAL b;
BYTE ch;
```

上面三个语句表示定义了三个变量，变量的定义形式都是合法的。其中，变量 a 的类型为 int 类型，变量 b 的类型为 float 类型，变量 ch 的类型为 unsigned char 类型。

在使用 typedef 定义类型名时，新的类型名命名要符合标识符命名规则。一般情况下，新的数

据类型名使用大写字母，用来提醒编程者这个类型名称实际上是一个数据类型名的缩写。当然，新的数据类型名字也可以使用小写字母，例如：

```
typedef int integer;
typedef float real;
typedef unsigned char byte;
```

上面这三条语句用 typedef 定义新数据类型名字的方法也是正确的。

typedef 不但可以为基本类型、指针类型定义新的类型说明符，也可以应用于结构体类型。例如：

```
typedef struct complex
{  float real;
   float imag;
} COMPLEX;
```

经过以上定义后，就可以用类型 COMPLEX 代替 struct complex 类型来表示复数。使用 typedef 的原因之一是，为经常出现的类型创建一个方便的、可识别的名称。

使用 typedef 来命名一个结构类型时，也可以省去结构的标记，例如：

```
typedef struct {double x; double y;} rect;
rect r1={3.0, 6.0};
rect r2;
r2=r1;
```

以上语句等价于：

```
typedef struct {double x; double y;} rect;
struct {double x; double y;} r1={3.0, 6.0};
struct {double x; double y;} r2;
r2=r1;
```

【例 9.12】使用 typedef 实现两个复数相加。

```
#include <stdio.h>
typedef struct complex
{  float real;
   float vir;
}Complex;
Complex Operation();
int main(void)
{  Complex p;
   p=Operation();
   printf("加减后的复数为：%2fi+%2f\n", p.real, p.vir);
   return 0;
}
Complex Operation()
{  float real1, real2;
   float virtual1, virtual2;
   Complex p;
   printf("请输入 real1 和 real2：");
   scanf("%f,%f", &real1, &real2);
   getchar();
   printf("请输入 virtual1 和 virtual2：");
   scanf("%f,%f", &virtual1, &virtual2);
   p.real=real1+real2;
```

```
    p.vir=virtual1+virtual2;
    return p;
}
```

运行结果如图 9-13 所示。

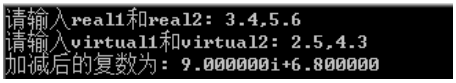


图 9-13　例 9.12 运行结果

【例 9.13】在主函数中输入结构体各个成员的值，在另一个函数中将其输出。

```
#include <stdio.h>
typedef struct
{  char name[14];
   int age;
   char addr[20];
}person;
void print(person *p)
{  printf("%-14s%-6s%-20s\n","Name","Age","Address");
   printf("%-14s%-6d%-20s\n",p->name,p->age,p->addr);
}
int main(void)
{  person s;
   printf("Name:\t");
   gets(s.name);
   printf("Age:\t");
   scanf("%d",&s.age);
   getchar();
   printf("Address:");
   gets(s.addr);
   print(&s);
   return 0;
}
```

运行结果如图 9-14 所示。

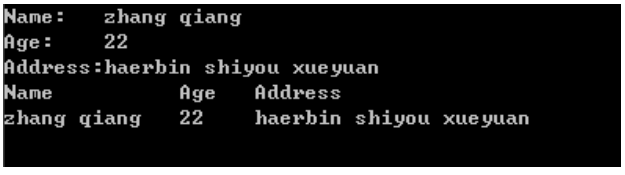


图 9-14　例 9.13 运行结果

9.8　程序设计举例

【例 9.14】已知某年的元旦是星期几，打印该年某一月份的日历表。

```
#include "stdio.h"
typedef struct
```

```
{  int year,month,day;
   enum weekday{sun,mon,tue,wed,thu,fri,sat} week;
}daily;

 void montable(daily d)
{  int i,s,ds;  daily md;
   md.year=d.year;md.day=1;
   printf("Which month?");
   scanf("%d",&md.month);                          /*查看当年哪月日历表*/
   for(s=0,i=1;i<=md.mon;i++)
{  switch(i)
   {  case 1: case 3: case 5: case 7: case 8: case 10: case 12:ds=31;
      break;
      case 2:
      ds=(md.year%4==0&&md.year%100!=0||md.year%400==0)?29:28;
          break;
      case 4: case 6: case 9: case 11:ds=30;
      }
      s+=ds;
   }
   s-=ds;
   md.week=(enum daily::weekday)((s+d.week)%7);/*计算该月 1 号是星期几，同时
                                                 计算该月天数存入 ds 中*/
   printf("-===%4d Year,%2d Month==--\n",md.year,md.mon);
   printf(".....................................\n");
   printf("%5s%5s%5s%5s%5s%5s%5s\n",
                         "Sun","Mon","Tue","Wed","Thu","Fri","Sat");
   printf(".....................................\n");
   for(i=0;i<md.week*5;i++) printf(" ");        /*计算该月 1 号的打印位置*/
       for(i=1;i<=ds;i++)
       {  printf("%5d",i);
            md.week=(enum daily::weekday)(md.week+1);
          if(md.week==(enum daily::weekday)7)
          {  md.week=(enum daily::weekday)0;printf("\n");}
                                                 /*超过一周换行打印*/
       }
   if(md.week!=0) printf("\n");
   printf(".....................................\n");
   }
main()
{  daily days;
   printf("Which year?");scanf("%d",&days.year);    /*哪年日历*/
   printf("year %4d,Month 1,day 1 is weekday?\n",days.year);
   printf("0-Sun,1-Mon,2-Tue,3-Wed,4-Thu,5-Fri,6-Sat:");
   scanf("%d",&days.week);                          /*当年元旦那天是星期几*/
   days.month=days.day=1;
   montable(days);
}
```

运行结果如图 9-15 所示。

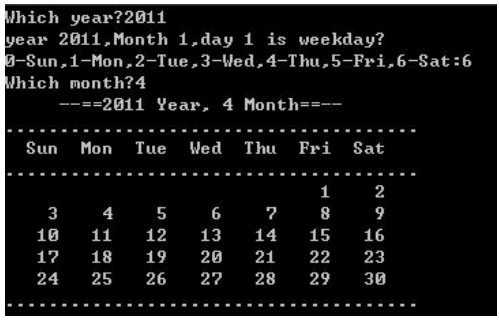


图 9-15　例 9.14 运行结果

读者可以就该程序考虑如何打印某年所有月份的日历表。若不知道元旦是星期几（可能仅知道当天是星期几），能否打印任何一年的日历表呢？另外，该程序中的枚举类型成员并没有真正按枚举量使用，怎么能真正用上呢？

【例 9.15】将两个带头结点的单链表连接成一个带头结点的单链表。

```
#include "stdio.h"
#include "malloc.h"
typedef struct lnode
{  int data;                          /*结点数据域*/
   struct lnode *next;                /*结点指针域*/
}lnode,*linklist;
linklist crelink(void)                /*建立单链表函数，返回结点指针类型*/
{  linklist p,q,head; int x;
   p=head=(linklist)malloc(sizeof(lnode));
                                      /*生成头结点，并使 p 指向该结点*/
   printf("Input node datas(-1=End):\n");
   scanf("%d",&x);
   while(x!=-1)                       /*以-1 作为结束条件*/
   {  q=(linklist)malloc(sizeof(lnode));
      q->data=x;
      p->next=q;                      /*连接 q 结点*/
      p=q;                            /*p 跳到 q 上，再准备连接下一个结点 q*/
      scanf("%d",&x);
   }
   p->next=NULL;                      /*置尾结点指针域为空指针*/
   return head;                       /*将已建立起来的单链表头指针返回*/
}
linklist concatlink(linklist h1,linklist h2)
{  linklist p=h1;
   while(p->next!=NULL) p=p->next;
      p->next=h2->next;
   free(h2);
   return h1;
   }
listhead(linklist head)               /*带头结点链表的输出*/
{  linklist p=head->next;             /*从第一个数据结点出发，依次输出*/
```

```
    printf("The linklist is:\n");
    while(p!=NULL)                          /*各结点的值，直到遇到 NULL*/
    {  printf("%5d",p->data);
       p=p->next;                           /*p 指针顺序后移一个结点*/
    }
    printf("\n");
}
main()
{  linklist h1,h2;
   h1=crelink();
   listhead(h1);
   h2=crelink();
   listhead(h2);
   h1=concatlink(h1,h2);
   listhead(h1);
}
```

运行结果如图 9-16 所示。

```
Input node datas(-1=End):
1 2 3 4 5 6 -1
The linklist is:
    1    2    3    4    5    6
Input node datas(-1=End):
0 9 8 7 -1
The linklist is:
    0    9    8    7
The linklist is:
    1    2    3    4    5    6    0    9    8    7
```

图 9-16　例 9.15 运行结果

本 章 小 结

C 语言的结构提供了在同一个数据对象中存储几个不同类型的数据项的方法。可以使用标记来代表一个具体的结构模板，并声明该结构类型的变量。使用成员点（.）运算符可以通过使用结构模板中的标签来访问结构的各个成员。

如果有一个指向结构的指针，可以使用该指针以及间接成员运算符(->)代替名字和点运算符来访问结构的各个成员。如果要得到结构的地址，可以使用运算符&。与数组不同的是，结构名并不是结构的地址。C 语言允许把结构作为参数传递，把结构作为返回值，并允许把一个结构赋值给另一个相同类型的结构。

联合类型使用与结构相同的语法，但联合成员共享一个公共的存储空间。联合存储选择成员列表中一个单独的数据项类型，而不像结构体类型同时存储多个数据类型。也就是说，如果一个结构体类型可以保存一个 int 型、一个 double 型、一个 char 型数据，那么相应的联合能保存一个 int 型，或者一个 double 型，或者一个 char 型的数据。

枚举类型可以创建一组代表整数常量的符号（枚举常量），也允许定义相关联的枚举类型。typedef 用来定义 C 语言标准类型的别名。一个函数的名称可以给出该函数的地址。这个指向函数的地址可以作为参数被传递给使用该函数的另一个函数。

习　题　9

一、单选题

1．下面对结构变量的叙述中错误的是（　　）。

A．相同类型的结构变量间可以相互赋值

B．通过结构变量，可以任意引用它的成员

C．结构变量中某个成员与这个成员类型相同的简单变量间可相互赋值

D．结构变量与简单变量间可以赋值

2．下列关于结构类型与结构变量的说法中，错误的是（　　）。

A．结构类型与结构变量是两个不同的概念，其区别如同 int 类型与 int 型变量的区别一样

B．结构体类型可将不同数据类型但相互关联的一组数据，组合成一个有机整体使用

C．结构类型名和数据项的命名规则，与变量名相同

D．结构类型中的成员名，不可以与程序中的变量同名

3．当定义一个结构体变量时，系统为它分配的内存空间是（　　）。

A．结构中一个成员所需的内存容量

B．结构中第一个成员所需的内存容量

C．结构体中占内存容量最大者所需的容量

D．结构中各成员所需内存容量之和

4．设有下列结构型变量 w 的定义，则表达式 sizeof(w)的值是（　　）。

```
struct
{  long num;
   char name[15];
   union
   {  float x;
      short z;
   }yz;
}w;
```

A．19　　B．20　　C．23　　D．25

5．设有以下说明语句：

```
struct ex
{  int x;
   float y;
   char z
}example;
```

则下面的叙述中不正确的是（　　）。

A．struct ex 是结构体类型　　B．example 是结构体类型名

C．x，y，z 都是结构体成员名　　D．struct 是结构体类型的关键字

6．若有如下结构类型定义：

```
struct bd
```

```
{  int x;
   float y;
}r,*p=&r;
```

则对 r 中的成员 x 的正确引用是（　）。

A．(*p).r.x　　B．(*p).x　　C．p->r.x　　D．p.r.x

7．若有如下结构类型定义以及有关的语句：

```
struct ms
{  int x;
   int *p;
}s1,s2;
s1.x=10;
s2.x=s1.x+10;
s1.p=&s2.x;
s2.p=&s1.x;
*s1.p+=*s2.p;
```

则执行以上语句后，s1.x 和 s2.x 的值应该是（　）。

A．10,30　　B．10,20　　C．20,20　　D．20,10

8．若定义以下结构体数组：

```
structc
{  int x;
   int y;
}s[2]={1,3,2,7};
```

则语句 printf("%d",s[0].x*s[1].x)的输出结果为（　）。

A．14　　B．6　　C．2　　D．21

9．运行下列程序段，输出结果是（　）。

```
struct country
{  int num;
   char name[10];
}x[5]={1,"China",2,"USA",3,"France",4,"England",5,"Spanish"};
struct country *p;
p=x+2;
printf("%d,%c",p->num,(*p).name[2]);
```

A．3,a　　B．4,g　　C．2,U　　D．5,S

10．根据下面的定义，能输出 Mary 的语句是（　）。

```
struct person
{  char name[9];
   int age;
};
struct personclass[5]={"John",17,"Paul",19,"Mary",18,"Adam",16};
```

A．printf("%s\n",class[1].name);

B．printf("%s\n",class[2].name);

C．printf("%s\n",class[3].name);

D．printf("%s\n",class[0].name);

11．运行下列程序，输出结果是（　）。

```
struct country
{  int num;
   char name[20];
}x[5]={1,"China",2,"USA",3,"France",4,"Englan",5,"Spanish"};
int main(void)
{  int i;
   for(i=3;i<5;i++)
        printf("%d%c",x[i].num,x[i].name[0]);
   return 0;
}
```

A．3F4E5S　　B．4E5S　　C．F4E　　D．c2U3F4E

12．有枚举型定义如下：enums {x1,x2=5,x3,x4=10}x;，则枚举变量 x 可取的枚举元素 x2、x3 所对应的整数常量值是（　）。

A．1,2　　B．2,3　　C．5,2　　D．5,6

13．在 TurboC 中有如下定义：

```
union dat
{  int i;
   char ch;
   float f;
}x;
```

x 在内存中占的字节数为（　）。

A．4　　B．7　　C．8　　D．6

14．若字符'0'的 ASCII 码的十进制数为 48，且数组的第 0 个元素在低位，则以下程序的输出结果是（　）。

```
#include<stdio.h>
int main()
{  union
   {  int i[2];
      long k;
      char c[4];
   }r,*s=&r;
   s->i[0]=0x39;
   s->i[1]=0x38;
   printf("%c\n",s->c[0]);
   return 0;
}
```

A．39　　B．9　　C．38　　D．8

15．以下选项中不能正确把 cl 定义成结构体变量的是（　）。

A．typedef struct
{ int red;

```
    int green;
    int blue;
} COLOR;
COLOR cl;
```

B．
```
struct color cl
{   int red;
    int green;
    int blue;
};
```

C．
```
struct color
{   int red;
    int green;
    int blue;
}cl;
```

D．
```
struct
{   int red;
    int green;
    int blue;
}c1;
```

16．以下叙述中错误的是（　　）。

A．可以通过 typedef 增加新的类型

B．可以用 typedef 将已存在的类型以一个新的名字来代表

C．用 typedef 定义新的类型名后，原有类型名仍有效

D．用 typedef 可以为各种类型起别名，但不能为变量起别名

17．以下各选项要求定义一种新的类型名，其中正确的是（　　）。

A．typedef v1 int;　　B．typedef v2=int;

C．typedef int v3;　　D．typedef v4:int;

18．若设有以下语句：

```
typedef struct S
{  int g;
   char h;
}T;
```

则下面叙述中正确的是（　　）。

A．可用 S 定义结构体变量　　B．可以用 T 定义结构体变量

C．S 是 struct 类型的变量　　D．T 是 struct S 类型的变量

二、判断题

1．结构体中的成员名不可以与程序中的变量名相同。（　　）

2．结构体类型只有一种。（　　）

3．在程序中定义了一个结构体类型后，可以多次用它来定义具有该类型的变量。（　　）

4．可以将一个结构体变量作为一个整体进行输入和输出。（　　）

5．结构体类型数据在内存中所占字节数不固定。（　）

6．在内存中存储结构体类型的变量要占连续一段的存储单元。（　）

7．用 typedef 可以声明各种类型名，也可以用来定义变量。（　）

8．被定义为指向某结构体类型数据的指针变量，既可以指向具有该类型的变量，又可以指向其中的一个成员。（　）

9．用 typedef 不仅对已经存在的类型增加一个类型名，而且还可以创造新的类型。（　）

10．一旦定义了某个结构体类型后，系统将为此类型的各个成员项分配内存单元。（　）

11．能在一个存储区内处理不同的类型的数据是共用体。（　）

12．一个共用体变量只能存放其中一个成员的值。（　）

13．共用体变量的各个成员所占内存单元是相同的，都从同一地址开始。（　）

14．共同体变量所占的内存长度等于最长的成员的长度。（　）

15．在定义枚举时，枚举常量可以是标识符或数字。（　）

三、程序填空题

1．有以下说明定义和语句，可用 a.day 引用结构体成员 day，请写出引用结构体成员 a.day 的其他两种形式_____和_____。

```
struct{int day;char mouth;int year;}a,*b;
b=&a;
```

2．以下程序用来输出结构体变量 stex 所占存储单元的字节数，请填空。

```
struct st
{  char name[20];
   double score;};
int main(void)
{  struct stex;
   printf("ex size: %d\n",sizeof(_____));
   return 0;
}
```

3．若有如下结构体说明：

```
struct STRU
{  int a, b;
   char c;
   doubled:
   struct STRU p1,p2;
};
```

请填空，以完成对 t 数组的定义，t 数组的每个元素为该结构体类型。

```
_____ t[20];
```

4．若有以下说明和定义语句，则变量 w 在内存中所占的字节数是_____。

```
union aa
{  float x, y;
   char c[6];
};
```

```
struct st
{  union aa v;
   float w[5];
   double ave;
} w;
```

5．以下程序的运行结果是_____。

```
# include <string.h>
typedef struct student
{  char name[10];
   long sno;
   float score;
}STU;
int main(void)
{  STU a={"zhangsan",2001,95};
   b={"Shangxian",2002,90};
   STU c={"Anhua",2003,95},d,*p=&d;
      d=a;
   if(strcmp(a.name,b.name)>0)
      d=b;
   if(strcmp(c.name,d.name)>0)
      d=c;
   printf("%ld%s\n",d.sno,p->name);
   return 0;
}
```

6．计算某日是当年的第几天。

```
#include <stdio.h>
struct
{  int year;
   int month;
   int day;
}data;
int main(void)
{  int days;
   printf("请输入日期(年、月、日)：");
   scanf("%d, %d, %d", &data.year, &data.month, &data.day);
   switch(data.month)
   {  case 1:days = data.day;
             break;
      case 2:days = data.day+________;
             break;
      case 3:days = data.day+59;
             break;
      case 4:days = data.day+90;
             break;
      case 5:days = data.day+________;
```

```
            break;
      case 6:days = data.day+151;
            break;
      case 7:days = data.day+181;
            break;
      case 8:days = data.day+212;
            break;
      case 9: days = data.day+243;
            break;
      case 10:days = data.day+273;
            break;
      case 11:days = data.day+304;
            break;
      case 12:days = data.day+334;
            break;
   }
   if(data.year%4==0&&data.year%100!=0_________data.year%400==0)
      if(data.month>=3)
         days =_________;
   printf("%d 月%d 日是%d 年的第%d 天.\n", data.month, data.day,data.year, days);
   return 0;
}
```

第 10 章　文 件 系 统

10.1　概　　述

文件（File）在计算机系统中的作用是很重要的。文件用来存放程序、文档、数据、信件、表格、图片和其他很多种类的信息。编写程序时，从文件读取信息或者将结果写入文件是一种经常性的需求。C 语言提供了功能强大的文件操作通信方法。在程序中访问文件需要首先打开文件，然后使用专门的 I/O（输入/输出）函数读取文件或者写入文件。

文件是程序设计中的一个重要概念。所谓文件一般是指存储在外部介质上的数据的集合。操作系统是以文件为单位对数据进行管理的，而文件是以文件名来标识的，每个文件都用唯一的文件名（主文件名.后缀格式）来标识。操作系统对文件实行按名存取，进行读出、写入等有关操作。比如，文件 stdio.h 就是一个包含一些有用信息的文件的名称。广义上，操作系统将每一个与主机相连的输入/输出设备都看成文件，例如，显示器、打印机是输出文件，键盘是输入文件。

本章的重点是掌握文件和文件指针的概念，以及文件的定义方法；了解文件打开和关闭的概念的方法；掌握有关文件的函数。本章的难点是掌握缓冲文件系统的使用。

10.2　文件类型和指针

根据文件的编码方式（数据的存储形式），可以将文件分为 ASCII 文件和二进制文件。ASCII 文件又称为文本文件或正文文件，它的每一个字节中存放一个 ASCII 数据，代表一个字符。二进制文件是把内存中的数据按其在内存中的存储形式原样输出到磁盘文件中存放的。例如，整型 1268 这个数据在内存中占 2 个字节存储，用二进制文件存储也占 2 个字节。如果用 ASCII 文件存储 1268，则要占 4 个字节，比用二进制形式存储要多占存储空间。但是，ASCII 文件在处理上比较方便，不需要进行中间的数据转换过程。1268 的二进制文件形式如图 10-1 所示。1268 的二进制文件形式如图 10-2 所示。

“1”的 ASCII 码	“2”的 ASCII 码	“6”的 ASCII 码	“8”的 ASCII 码
00110001	00110010	00110110	00111000

图 10-1　1268 的文本文件形式

00000100	11110100

图 10-2　1268 的二进制文件形式

在缓冲文件系统中，有个关键概念就是文件类型指针，简称文件指针。每个被使用的文件都在内存中开辟一个区域，用来存放文件的有关信息。这些信息保存在一个由系统定义的名为 FILE 结构体类型的变量中。FILE 是一个标准的标识符，但不是关键字。系统定义的结构体 FILE 类型包含成员如下：

```
typedef struct
{  short level;                  /*缓冲区满或空的程度*/
   unsigned flags;               /*文件状态标志*/
   char fd;                      /*文件描述符*/
```

```
    unsigned char hold;          /*如无缓冲区不读取字符*/
    short bsize;                 /*缓冲区的大小*/
    unsigned char *buffer;       /*数据缓冲区的位置*/
    unsigned char *curp;         /*指针当前的指向*/
    unsigned istemp;             /*临时文件指示器*/
    short token;                 /*用于有效性检查*/
} FILE;                          /*自定义文件类型名 FILE*/
```

不同的 C 语言编译系统的结构体 FILE 类型所包含的内容不完全相同，但大同小异。

对文件进行访问时，标准的操作包括四个基本步骤：

（1）定义的一个文件类型的指针。

（2）打开要进行操作的文件。

（3）通过文件读/写函数，对文件进行读或写。

（4）关闭已打开的文件。

从上面的操作步骤可以看出，对文件进行操作首先需要定义一个文件指针。在下面的程序中，将用这个文件类型指针指向需要操作的文件，定义结构体类型的指针变量的格式为：

```
FILE *指针变量标识符;
```

例如：

```
FILE *fp;
```

这里指针 fp 就是一个将要指向 FILE 类型结构体的指针变量，通过该结构体变量中的文件信息能够访问该文件。FILE 文件类型的定义说明存放在头文件 stdio.h 中，所以在进行文件操作时一定要包含 stdio.h 这个头文件。如果有 N 个文件，一般也应该设置 N 个指针变量，使它们分别指向 N 个文件，以便可以实现对各个文件的访问。

10.3　文件的打开与关闭

对文件进行读/写操作时，需要先打开文件，在使用结束时，关闭文件。

所谓的打开文件就是指将文件信息从磁盘装入计算机内存，建立文件的各种有关信息，并使文件指针指向该文件，即建立文件类型指针与文件之间的关联。所谓的关闭是指撤销文件信息区和文件缓冲区，使文件类型指针不再指向该文件，即切断已建立的文件类型指针与文件之间的关联。在对文件执行关闭操作后，将不能再对文件进行读/写，以保证文件内存储信息的安全性。

10.3.1　文件的打开函数（fopen()函数）

C 语言使用函数 fopen()（file open）实现打开文件。这一函数在头文件 stdio.h 中声明定义。fopen()函数的调用方式为：

```
fopen(文件名，打开文件的方式);
```

fopen()函数的第一个参数是要打开文件的文件名。fopen()函数的第二个参数是用来指定打开文件方式的字符。如果由文件名指定的文件成功打开，则返回一个指向打开文件的文件类型指针，否则返回 NULL。

例如：

```
fopen("file1.dat", "r");
```

上面语句表示要打开名为 file1.dat 的文件，打开文件方式为只读，关于文件打开方式，将在下面介绍。如果文件 file1.dat 成功地被打开，则返回一个指向文件 file1.dat 的文件指针，否则返回 NULL。为了下面对文件进行读/写，fopen()函数的返回值应赋给一个 FILE 指针变量，否则，fopen()函数的返回值就会丢失，将无法对 file1.dat 文件进行其他操作。完整地打开文件 file1.dat 的语句代码为：

```
FILE *fp;                              //fp 为文件类型指针
fp=fopen("file1.dat", "r");            //文件指针 fp 指向 file1.dat
```

将 fopen()函数的返回值指向文件 file1.dat，文件的指针赋给了已定义的文件指针 fp，使得指针 fp 和文件 file1.dat 相联系，即使 fp 指向文件 file1.dat。在打开文件的同时，还告知编译系统使用文件的方式。使用文件的方式如表 10-1 所示。

表 10-1 使用文件的方式

文件的使用方式	含义	文件的打开方式	含义
r（只读文本）	为输入打开一个文本文件	r+（读/写文本）	为读/写打开一个文本文件
w（只写文本）	为输出打开一个文本文件	w+（读/写文本）	为读/写建立一个新的文本文件
a（追加文本）	向文本文件尾部追加数据	a+（读/写文本）	为读/写打开一个文本文件
rb（只读二进制）	为输入打开一个二进制文件	rb+（读/写二进制）	为读/写打开一个二进制文件
wb（只写二进制）	为输出打开一个二进制文件	wb+（读/写二进制）	为读/写建立一个新的二进制文件
ab（追加二进制）	向二进制文件尾部追加数据	ab+（读/写二进制）	为读/写打开一个二进制文件

（1）“r”表示只读（read only）方式。用这种方式打开的文件只能从文件读出数据，而且要求要打开的文件必须存在，并且存有数据。否则，将会出错。

（2）“w”表示只写（write only）方式。用这种方式打开的文件只能向该文件写数据。用“w”方式打开的文件，无论文件是否存在，都会重建该文件。如果不存在，将会在指定位置建立一个与 fopen()函数中指定的同名的文件。若存在则先删除原文件，再新建一个同名文件。

（3）“a”表示追加（append）方式。当希望在文件末尾添加数据时，应该使用“a”方式。以“a”方式打开的文件，文件的读/写位置标记位于文件末尾，添加的数据也将会加入到文件末尾。用“a”方式打开的文件，如果文件不存在，则建立该文件，否则，将文件中的位置指针移到文件末尾，准备追加数据。

（4）“+”表示可读、可写。当文件使用方式中包含“+”时，表示对打开的文件既可以进行读操作，也可以进行写操作。

（5）如果不能实现打开操作，fopen()函数将返回一个出错信息。出错原因有很多，可能是用 r 方式打开一个根本不存在的文件、磁盘出故障、磁盘已满、无法新建文件，等等。

常用如下代码打开文件并测试返回值。

```
if((fp=fopen("file1.dat","r")==NULL)
{  printf("Cannot open this file!\n");
   exit(0);
}
```

上面的代码表示，在打开文件时，将检查打开文件操作是否出错。如果出错，fopen()函数返回 NULL，则在输出终端上输出“Cannot open this file!”，并关闭所有文件，退出正在执行的程序。

10.3.2　文件关闭函数（fclose()函数）

C 语言使用 fclose()函数关闭文件。fclose()函数的调用方式为：

```
fclose(fp);
```

fclose()函数关闭由指针 fp 指定的文件，同时根据需要刷新缓冲区。如果指针 fp 指定的文件成功关闭，fclose()函数将返回值 0，否则返回 EOF。EOF 是文件结束标志，有时也作为文件操作出错的标志。EOF 是在头文件 stdio.h 中定义的一个宏：

```
#define EOF -1;
```

EOF 是一个常量，其值为–1。

更为完整的程序还要检查文件是否成功关闭。磁盘已满、磁盘被移走或者出现 I/O 错误等都会导致 fclose()函数执行失败。常用下面的选择语句判断文件是否正常关闭。

```
if(fclose(fp)!=0)
   printf("Error in closing file\n");
```

10.4　文件的读/写

10.4.1　字符读/写函数（fgetc()函数和 fputc()函数）

文件打开后，就可以对文件进行读/写操作。C 语言使用 fgetc()函数从文件指针指向的文件当前位置读出一个字符，然后文件位置指针自动后移，指向文件中的下一个字符。fgetc()函数调用格式为：

```
fgetc(fp);
```

fgetc()函数从文件指针 fp 指向的文件当前位置读出一个字符，如果读入成功，则返回值为读入的字符。一般会将读出的字符传递给一个字符变量。例如：

```
ch=fgetc(fp);
```

在头文件 stdio.h 中有如下的宏定义：

```
#define getc(fp) fgetc(fp)
#define getchar() fgetc(stdin)
```

从上面的宏定义看出，也可以使用 getc()函数或者 getchar()函数来进行读字符的操作。

程序从文件中读取数据时，需要在到达文件结尾时停止。如果在尝试读入字符时发现已经达到文件结尾，遇到文件结束符，则返回结束符 EOF。为了避免读取空文件可能带来的问题，程序应该在进入循环体之前尝试进行第一次读取。所以，应该对文件输入使用 while 循环，而不是 do…while 循环。

【例 10.1】将磁盘文件 wacky.txt 的信息读出，并显示到屏幕上。

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{  char ch;                          //字符变量 ch 判断是否到达文件末尾
   FILE *fp;
```

```
    if((fp=fopen("wacky.txt","r"))==NULL)
    {   printf("File notexist!\n");
        exit(1);
    }
    ch=getc(fp);                    //获取第一个文件字符
    while(ch!=EOF)
    {  putchar (ch);                //将文件字符输出
       ch=fgetc(fp);                //获取下一个文件字符
    }
    fclose(fp);
    printf("\n");
    return 0;
}
```

运行结果请查看文件 wacky.txt。文件 wacky.txt 与程序在同一文件夹下。

对例 10.1 中循环读入字符的代码还可以进一步改进简化。

【例 10.2】将磁盘文件 wacky.txt 的信息读出并显示到屏幕上。

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{  char ch;
   FILE *fp;
   if((fp=fopen("wacky.txt","r"))==NULL)
   {  printf("\nFile not exist!");
      exit(1);
   }
   while((ch=fgetc(fp))!=EOF)
      putchar(ch);
   fclose(fp);
   return 0;
}
```

运行结果请查看文件 wacky.txt。

表达式(c=fgetc(fp)!=EOF 用来判断使用 fgetc()函数读到的字符是否为文件结束字符 EOF。因为 fgetc()函数是 while 语句判断条件的一部分，所以将在进入循环体之前执行该语句。应该避免下面的形式：

```
while(ch!=EOF)                  //在首次使用 ch 时，其值尚未确定
{  ch=fgetc(fp);
   putchar(ch);
}
```

上面提到的常量 EOF，是文本文件的测试标志，其值为–1。因为文本文件中都是字符的 ASCII，所以没有–1 这个数值。但是，不能用它来测试一个二进制文件的结束与否，因为二进制文件中允许含有–1 这个数值。除了使用 EOF 外，还可以用 feof()函数来得到文件的结束。其调用格式如下：

```
feof(指向文件指针);
```

可以把上面程序中的程序段：

```
while((ch=fgetc(fp))!=EOF)
  putchar(ch);
```

改成：

```
while(!feof(fp))
  putchar(fgetc(fp));
```

程序修改后，运行结果是完全相同的。

与 fgetc()函数相对的字符写入函数是 fputc()函数。fputc()函数可以把内存中的一个字符写入到指定的磁盘文件中去。fputc()函数调用格式为：

```
fputc(ch,fp);
```

其中，ch 表示要写入文件的字符。fputc()函数将把一个字符 ch 写到 fp 指向的磁盘文件中。如果 fputc()函数写入字符成功，则返回这个输出的字符，反之输出失败，则返回一个 EOF，这里 EOF 表示文件操作出错。

在头文件 stdio.h 中有如下的宏定义：

```
#define putc(ch,fp) fputc(ch,fp)
#define putchar(ch) fputc(ch,stdout)
```

所以，也可以使用 putc()函数或者 putchar()函数来进行写字符的操作。

【例 10.3】将文件 filea.dat 的内容复制到文件 fileb.dat 中。

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{  FILE *f1,*f2;
   char c;
   if((f1=fopen("filea.dat" , "r"))==NULL)
   {  printf("File cannot open!\n");
      exit(1);
   }
   if((f2=fopen("fileb.dat","w"))==NULL)
   {  printf("File cannot creat! \n");
      exit(1);
   }
   while((c=fgetc(f1))!=EOF)
      fputc(c,f2);
   fclose(f1);
   fclose(f2);
   return 0;
}
```

运行结果请查看文件 fileb.dat。

在例 10.3 中首先打开已经存在的文件 filea.dat，以写入的方式打开文件 fileb.dat。在 while 循环中，每次循环利用 fgetc()函数读出一个字符，再利用 fputc()函数将其写入到文件指针 f2 指向的文件 fileb.dat 中。

10.4.2 字符串读/写函数（fgets()函数和 fputs()函数）

除可以采用单个字符的方式对文件进行读、写操作外，还可采用以字符串为单位的方式对文件进行读或写。

fgets()函数是读字符串函数，其调用方式为：

```
fgets(buffer,len,fp);
```

fgets()函数将从指针 fp 指向的文件中读取 len-1 个字符，并把这些字符送到由指针 buffer 指向的字符数组中。如果读入指针 fp 指向的文件成功，则返回指针 buffer 的首地址，否则返回空指针 NULL。

fgets()函数需要接收 3 个参数。第一个参数 buffer 指针与 gets()函数一样，用于指定读出的字符串的存放位置。第二个参数 len 为整数，表示读出字符串的最大长度。第三个参数是文件指针 fp 指向要读取的文件。

其中，由 buffer 指针指向的用来存放字符数组的长度应该大于 len，可存放 len-1 个字符，这里字符数组的最后一个元素为'\0'。当 fgets()函数读取到比字符串的最大长度少一个的字符时，或者在没有读到字符最大数目前，读取到第一个换行字符“\n”，或者读取到文件结尾 EOF 结束符号时，将结束读入操作，fgets()函数向末尾添加一个 NULL 字符以构成一个字符串。所以，字符串的最大长度代表字符的最大数目再加上一个 NULL 字符。

【例 10.4】利用 fgets()函数，将文本文件 filea.txt 中的内容全部读出并显示在屏幕上。

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{  FILE *fp;
   char str[81];
   if((fp=fopen("filea.txt","r"))==NULL)
   {  printf("Cannot open file!");
      exit(1);
   }
   while(fgets(str,81,fp)!=NULL)
      puts(str);
   fclose(fp);
}
```

运行结果请查看文件 filea.txt。

文本文件一般每行最多 80 个字符，再加上一个行结束符号换行符'\n'，则每行至少需要 81 个字节的存储空间。例 10.4 中首先定义了文件指针 fp，并将其指向文本文件 filea.txt。fopen()函数以只读方式打开文件。在 while 循环中，每次循环读入 80 个字符，利用 puts()函数将其输入到显示器上，直到文件结束。

与 fgets()函数相对应的是 fputs()函数。fputs()函数是写字符串函数，其调用格式为：

```
fputs(buffer,fp);
```

fputs()函数把由指针 buffer 指向的字符数组中的字符串写入由文件指针 fp 指向的文件中。fputs()函数需要两个参数，分别是字符串地址 buffer 和文件指针 fp。其中，指针 buffer 可以是字符串常量、字符数组名、指针变量。

【例 10.5】从键盘输入若干行字符，将它们添加到磁盘文件 filea.txt 中。

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{  FILE *fp;
   char buf[81];
   if((fp=fopen("filea.txt","a"))==NULL)
   {  printf("File cannot open!");
      exit(1);
   }
   while(fgets(buf,81,stdin)!=NULL)
      fputs(buf,fp);
   fclose(fp);
   return 0;
}
```

运行结果请查看文件 filea.txt。

10.4.3　数据块读/写函数（fread()函数和 fwrite()函数）

前面介绍了可以以单个字符为单位对文件进行读/写，也可以指定字符长度以字符串形式进行文件读/写。C 语言提供了另外一种读/写的方法，即以数据或者一组数据为单位对一个数据字块进行读/写操作。

fread()函数是数据块读函数，其调用格式为：

```
fread(buffer,size,count,fp);
```

fread()函数从文件指针 fp 所指向的文件中读出 count 个 size 大小的数据到内存空间 buffer 上。如果读入成功则返回 count 值，否则返回非 count 值。其中，buffer 是读入数据的存放地址，size 是要读的字节数，count 指出要读多少个 size 字节的数据项，fp 是文件指针。

fwrite()函数是数据块写函数，其调用格式为：

```
fwrite(buffer,size,count,fp);
```

fwrite()函数将地址 buffer 中的 count 个 size 大小的数据写入到文件 fp 所指向的文件中。其中，buffer 是要输出数据的起始地址。其余参数与 fread()函数参数的含义相同。例如：

```
float a[2];
a[0]=1.0;
a[1]=2.0;
fwrite(a,4,2,fp);
```

这个函数将数组 a 中存放的两个数据写入到文件指针 fp 所指向的文件。其中，a 是一个 float 型数组名。一个 float 型数据占 4 个字节。

【例 10.6】从键盘输入两个学生的信息，写入磁盘文件 student.dat 中。

```
#include <stdio.h>
#define size sizeof(struct student)
struct student
{  char name[10];
   int num;
```

```
    int age;
    char addr[15];
    }boy[2],*pp;
int main(void)
{  FILE *fp;
    char ch,filename[20];
    int i;
    gets(filename);                    /*输入文件名*/
    fp=fopen(filename, "w");           /*以读/写方式打开二进制文件*/
    printf("\ninput data\n");
    pp=boy;
    for(i=0;i<2;i++,pp++)              /*输入两个学生的信息*/
       scanf("%s%d%d%s",pp->name,&pp->num,&pp->age,pp->addr);
    fwrite(boy,size,2,fp);
    fclose(fp);
    return 0;
}
```

运行结果请查看文件。

在例 10.6 中，首先定义学生信息的结构体 student，同时定义了 student 类型的数组 boy，使用 fopen()函数以读/写的方式打开指定文件，在 for 循环中利用标准输入 scanf()函数给 boy 数组元素赋值。最后使用 fwrite()函数将数组 boy 中的数据写入文件指针 fp 所指向的文件中。在程序结束前，将文件关闭。

10.4.4 格式化读/写函数（fscanf()函数和 fprintf()函数）

标准输入函数 scanf()可以从键盘获得数据，标准输出函数 printf()可以向显示器或打印机输出数据。对于文件中的数据，C 语言提供了专门的 fprintf()函数和 fscanf()函数进行文件数据的格式化输入和输出。

fprintf()函数可以按照格式字符串指定格式，将输出表列内容按照格式字符串指定的格式写入到文件指针所指定的文件中。其调用格式为：

```
fprintf(文件指针,格式字符串,输出表列);
```

例如：fprintf(fp, "%d,%6.2f",i,t);

上面的语句可以将整型变量 i 和实型变量 t 的值按%d 和%6.2f 的格式输出到文件指针 fp 所指向的文件中。

特别是当 fp 取为 stdout 时，fprintf()与 printf()相同。

例如：fprintf(stdout, "%s",s);

fprintf(stdout,"%s",s)等价于 printf("%s",s)。

fscanf()函数可以按照格式字符串指定格式，将表达式表中的数据存入到文件指针 fp 所指向的文件中。其调用格式为：

```
fscanf(文件指针,格式字符串,输入表列);
```

例如：fscanf(fp, "%d,%f",&i,&t);

上面的语句可以按格式字符串规定的格式，从文件指针 fp 指定的文件中读取数据分别送入变量 i 和 t 中。

特别是当 fp 取为 stdout 时，fscanf()与 scanf()相同。

例如：fscanf(stdio, "%s",s);

fscanf(stdio, "%s",s)等价于 scanf("%s",s)。

【例 10.7】fprintf()函数、fscanf()函数的使用。

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 40
int main(void)
{  FILE *fp;
   char words[MAX];
   if((fp=fopen("filea","a+"))= =NULL)
   {  fprintf (stdout, "Can't open \"words\" file.\n");
      exit (1);
   }
   puts("Enter words to add to the file, ");
   puts("press the Enter key at the beginning of a line to terminate.");
   while(gets(words)!=NULL&&words[0]!='\0')
      fprintf(fp,"%s",words);
   puts("File contents: ");
   rewind(fp);                          //回到文件的开始处
   while(fscanf(fp,"%s",words)= =1)
      puts (words);
   if(fclose(fp)!=0)
      fprintf (stderr, "Error closing file.\n");
   return 0;
}
```

运行结果如图 10-3 所示。假设文件 filea 中已存储内容为例 10.7 介绍使用函数。

```
Enter words to add to the file,
press the Enter key at the beginning of a line to terminate.
fscanf(),fprintf()

File contents:
例10-7介绍使用函数fscanf(),fprintf()
```

图 10-3　例 10.7 运行结果

通过该程序可以向 words 文件中加入单词。在 fopen("words","a+")语句中，会先创建一个文件名为 filea 的文件。如果文件不存在，则创建该文件。a+模式使得程序后面可以对文件 filea 进行追加单词的操作。这里用到文件定位函数 rewind()，将在 10.5 节中详细介绍。该命令使文件指针 fp 指向文件开始处，以保证在最后的 while 循环中可以打印文件的全部内容。当输入一个空行时，函数 gets()将数组的第一个元素 words[0]置为空字符，据此来终止追加字符输入。

【例 10.8】将 100 以内的所有素数写入磁盘文件 sushu.txt 中，并计算磁盘文件 sushu.txt 中的所有素数之和。

```
#include "stdio.h"
int main(void)
{  FILE *fp;
   int i,m,s=0;
```

```
    fp=fopen("sushu.txt","w+");
    for(m=2;m<=100;m++)
    {  for(i=2;i<=m/2;i++)
          if(m%i==0)  break;
       if(i>m/2)
          fprintf(fp,"%4d",m);
    }
    rewind(fp);
    while(fscanf(fp,"%d",&m)!=-1)
       s+=m;
    printf("s=%d\n",s);
    fclose(fp);
    return 0;
}
```

```
s=1060
```

运行结果如图 10-4 所示。

图 10-4　例 10.8 运行结果

【例 10.9】将一组学生记录以文件名 stufile.txt 形式存盘，其中每个学生记录包含学号、姓名、计算机、英语、总分等项，再将 stufile.txt 文件中的记录按总分降序排列，将总分相同的记录按学号升序排列后，存入到文件 stusort.txt 中。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define N 10
typedef struct
{  char num[7];
   char name[9];
   int com,eng,total;
}student;
int main(void)
{  int i,j;
   FILE *fp;
   student stu[N]={ {"202149","王学海", 83,77,0},
                    {"203120","刘玉芳",89,71,0},
                    {"201034","邱玲",76,69,0},
                    {"202062","郑玉梁",63,82,0},
                    {"201458","吕文斌",79,57,0},
                    {"201841","赵成",53,61,0},
                    {"202621","李可心",92,68,0},
                    {"203117","刘宁",76,60,0},
                    {"203302","张晨曦",71,65,0},
                    {"202512","钱宇航",88,90,0}};
   student t;
   fp=fopen("stufile.txt","w");
   for(i=0;i<N;i++)
   {  stu[i].total=stu[i].com+stu[i].eng;
      fprintf(fp,"%s\t%s\t%d\t%d\t%d\n",
           stu[i].num,stu[i].name,stu[i].com,stu[i].eng,stu[i].total);
   }
   printf("文件 stufile.txt 的内容是：\n");
```

```
    for(i=0;i<N;i++)
      printf("%s\t%s\t%d\t\t%d\t\t%d\n",
           stu[i].num,stu[i].name,stu[i].com,stu[i].eng,stu[i].total);
    printf("\n");
    printf("\n");
    fclose(fp);
    fp=fopen("stufile.txt","r");
      if((fscanf(fp,"%s\t%s\t%d\t%d\t%d\n",
        stu[0].num,stu[0].name,&stu[0].com,&stu[0].eng,&stu[0].total))==-1)
          exit(0);
    i=1;
    while((fscanf(fp,"%s\t%s\t%d\t%d\t%d\n",
        stu[i].num,stu[i].name,&stu[i].com,&stu[i].eng,&stu[i].total))!=-1)
    {  for(j=i-1,t=stu[i];j>=0&&t.total>stu[j].total;j--)
           stu[j+1]=stu[j];
      while(t.total==stu[j].total&&strcmp(t.num,stu[j].num)<0)
           stu[j+1]=stu[j--];
      stu[j+1]=t;
      i++;
    }
    fclose(fp);
    fp=fopen("stusort.txt","w");
      for(j=0;j<i;j++)
    fprintf(fp,"%s\t%s\t%d\t%d\t%d\n",
      stu[j].num,stu[j].name,stu[j].com,stu[j].eng,stu[j].total);
    fclose(fp);
    printf("文件 stusort.txt 的内容是: \n");
    for(i=0;i<N;i++)
      printf("%s\t%s\t%d\t\t%d\t\t%d\n",
        stu[i].num,stu[i].name,stu[i].com,stu[i].eng,stu[i].total);
    fclose(fp);
    return 0;
}
```

运行结果如图 10-5 所示。

```
文件stufile.txt的内容是:
202149  王学海  83              77              160
203120  刘玉芳  89              71              160
201034  邱玲    76              69              145
202062  郑玉梁  63              82              145
201458  吕文斌  79              57              136
201841  赵成    53              61              114
202621  李可心  92              68              160
203117  刘宁    76              60              136
203302  张晨曦  71              65              136
202512  钱宇航  88              90              178


文件stusort.txt的内容是:
202512  钱宇航  88              90              178
202149  王学海  83              77              160
202621  李可心  92              68              160
203120  刘玉芳  89              71              160
201034  邱玲    76              69              145
202062  郑玉梁  63              82              145
201458  吕文斌  79              57              136
203117  刘宁    76              60              136
203302  张晨曦  71              65              136
201841  赵成    53              61              114
```

图 10-5　例 10.9 运行结果

运行结果也可参考文件 stufile.txt、stusort.txt。

10.5 文件的定位（rewind()函数和 fseek()函数）

在每个打开的文件中，都有一个文件位置指针，它指向将要读/写的字符位置。当对文件进行顺序读/写时，每读完一个字符，该位置指针就自动移到下一个字符位置。实际上，常要求读/写文件中某些指定的部分。为了避免不必要的读或写的操作，可先移动文件的位置指针到需要读/写的位置，再进行读/写，这种读/写操作方式称为随机读/写。移动文件位置指针的操作称为文件的定位。文件定位操作是通过库函数的调用来完成的。

rewind()函数可以使文件指针 fp 指向文件的开始处。其调用格式为：

```
rewind(fp);
```

更为灵活的定位函数是 fseek()函数。fseek()函数可以实现文件的随机定位，即可以将文件指针移动到指定位置，再进行读/写。其调用格式为：

```
fseek(fp,offset,pos);
```

fseek()函数功能是，将指向文件的内部位置指针 fp 从 pos 指定的位置开始移动 offset 个字节。在 fseek()函数的 3 个参数中，第一个参数是一个指向被搜索文件的 FILE 文件指针，应该是已经使用 fopen()函数打开该文件。fseek()函数的第二个参数称为偏移量(offset)，表示从起始点开始要移动的距离。这个参数必须是一个 long 整型，可以为正（前移）、负（后移），也可以为零（保持不动）。第三个参数是 pos 移动的起始点（起始点模式参见表 10-2）。所以，起始点 pos 和相对偏移量 offset 共同指定了文件指针 fp 转移后的位置。在头文件 stdio.h 中指定了如表 10-2 所示的模式常量。

表 10-2 文件位置描述符

起始点	常量名标识	数字表示
文件开始	SEEK_SET	0
文件当前位置	SEEK_CUR	1
文件末尾	SEEK_END	2

例如：

```
fseek(fp,60L,SEEK_CUR);     //文件指针从当前位置向后移动 60 个字节
fseek(fp,-10L,SEEK_END);    //文件指针从文件末尾处向前移动 10 个字节
```

【例 10.10】fseek()函数使用举例。

```
#include <stdio.h>
#define N 5
typedef struct student
{  long sno;
   char name[10];
   float score[3];
}STU;
void fun(char *filename,STU n)
{  FILE *fp;
   fp=fopen(filename, "rb+");
```

```
    fseek(fp,-1L*sizeof(STU),SEEK_END);
    fwrite(&n,sizeof(STU),1,fp);
    fclose(fp);
}
void main(void)
{  int i,j;
   FILE *fp;
   STU ss[N];
   STU t[N]={{10001,"MaChao",91,92,77},
             {10002,"CaoKai",75, 60,88},
             {10003,"LiSi",85,70,78},
             {10004,"FangFang",90,82,87},
             {10005,"ZhangSan",95,80,88}};
   STU n={10006,"ZhaoYi",55,70,68};
   fp=fopen("student.dat","wb");
   fwrite(t,sizeof(STU),N,fp);
   fclose(fp);
   fp=fopen("student.dat","rb");
   fread(ss,sizeof(STU),N,fp);
   fclose(fp);
   printf("\nThe original data :\n\n");
   for (j=0; j<N; j++)
   {  printf("\nNo:%ld Name:%-8s Scores:",ss[j].sno, ss[j].name);
      for (i=0; i<3; i++)
         printf("%6.2f ",ss[j].score[i]);
      printf("\n");
   }
   fun("student.dat",n);
   printf("\nThedataaftermodifing:\n\n");
   fp=fopen("student.dat","rb");
   fread(ss,sizeof(STU),N,fp);
   fclose(fp);
   for(j=0;j<N;j++)
   {
      printf("\nNo:%ldName:%-8sScores:",ss[j].sno,ss[j].name);
      for(i=0;i<3;i++)
         printf("%6.2f",ss[j].score[i]);
      printf("\n");
   }
}
```

运行结果如图 10-6 所示。

ftell()函数可以返回一个文件的当前位置，这个位置是一个 long 类型的值。其调用格式为：

```
ftell(fp);
```

利用这个函数，可以测试一个文件所占的字节数。例如：

```
fseek(fp,0L,2);                    //将文件位置指针移到文件末尾
volume=ftell(fp);                  //测试文件尾到文件头的位移量
```

```
The original data :

No:10001 Name:MaChao   Scores: 91.00  92.00  77.00
No:10002 Name:CaoKai   Scores: 75.00  60.00  88.00
No:10003 Name:LiSi     Scores: 85.00  70.00  78.00
No:10004 Name:FangFang Scores: 90.00  82.00  87.00
No:10005 Name:ZhangSan Scores: 95.00  80.00  88.00
The data after modifing :

No:10001 Name:MaChao   Scores: 91.00  92.00  77.00
No:10002 Name:CaoKai   Scores: 75.00  60.00  88.00
No:10003 Name:LiSi     Scores: 85.00  70.00  78.00
No:10004 Name:FangFang Scores: 90.00  82.00  87.00
No:10006 Name:ZhaoYi   Scores: 55.00  70.00  68.00
```

图 10-6　例 10.10 运行结果

【例 10.11】利用 ftell()函数，反序显示一个文件。

```
#include <stdio.h>
#include <stdlib.h>
#define CNTL_Z  '\032'              //DOS 文本中的文件结尾标记
#define SLEN 50
int main(void)
{  char file[SLEN];
   char ch;
   FILE *fp;
   long count,last;
   puts("Enter the name of the file to be processed: ");
   gets(file);
   if((fp=fopen(file,"rb"))= =NULL)
   {  printf("reverse can't open %s\n", file);
      exit(1);
   }
   printf("\nthe originanl data of %s is\n", file);
   while(!feof(fp))
      putchar(fgetc(fp));
   fseek(fp, 0L,SEEK_SET);          //定位在文件开头处
   last=ftell(fp);
   printf("\n");
   printf("\n");
   fseek(fp,0L,SEEK_END);           //定位在文件结尾处
   last=ftell(fp);
   printf("the reverse data of %s is",file);
   for(count=1L;count <= last;count++)
   {  fseek(fp, -count, SEEK_END);
      ch=getc(fp);
      if(ch!=CNTL_Z&&ch!= '\r')
```

```
            putchar(ch);
        }
        putchar('\n');
        fclose(fp);
        return 0;
    }
```

运行结果如图 10-7 所示。

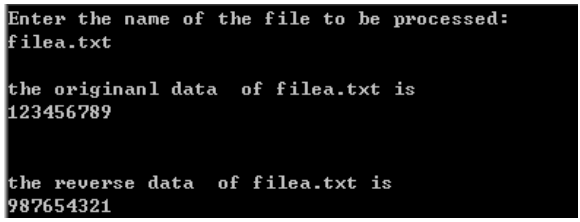


图 10-7　例 10.11 运行结果

在例 10.11 中，fseek(fp,0L,SEEK_END)语句把当前位置设置为从文件结尾处偏移 0 字节处，也就是将位置设定在文件结尾。接下来的语句 last=ftell(fp)把从文件开始到文件结尾的字节数目赋给 last。for 循环中的循环体语句如下：

```
fseek(fp,--count,SEEK_END);
ch=gete(fp);
```

在第一次执行时，将程序定位到文件结尾前的第一个字符，也即文件的最后一个字符，紧接着打印这个字符。在下一次循环中，再将文件指针定位到倒数第二个字符并打印，这种操作会一直持续到到达第一个字符。

10.6　文件错误处理函数（ferror()函数和 clearerr()函数）

ferror()函数用来测试文件操作的一些函数是否出现错误。如果发生读/写错误，ferror()函数将返回一个非零值，否则返回零值，其调用格式为：

```
ferror(fp);
```

前面根据返回值就可以测试该函数对文件操作是否出错。而 ferror()是一个统一的测试函数，用起来较为方便。值得注意的是，在一个文件的每一次函数调用中都会产生一个新的 ferror()函数值，因此，只有当前的测试是最有效的。在执行 fopen()函数时，自动将 ferror()函数的初始值置为 0。

clearerr()函数是清除错误标志函数，其调用格式为：

```
clearerr(fp)
```

clearerr()函数将文件出错标志置为 0，没有返回值。在程序运行时，只要文件操作出现错误，错误标志就一直保留，直到对同一文件调用 clearerr()、rewind()或任何其他一个输入/输出函数。

10.7　程序设计举例

【例 10.12】利用命令行参数完成 DOS 内部命令 COPY 的功能。

DOS 的 COPY 功能是将任意一个磁盘文件复制到指定的文件上。

1．完成键盘输入复制功能

```
#include "stdio.h"
include "stdlib.h"
main(int argc,char *argv[])
{  FILE *fp;
   char ch;
   if(argc!=2)
   {  printf("Input arguments error!\n");
      exit(0);
   }
   fp=fopen(argv[1],"w");
   while((ch=getchar())!='#')
      fputc(ch,fp);
   fclose(fp);
}
```

假设该程序以文件名“CCOPY.C”存盘，经过编译连接生成可执行文件“CCOPY.EXE”。“#”号之前的内容被存入文本文件“abc.txt”中。
运行方法参看图 10-8，结果参看 abc.txt。

```
C:\Users\Administrator>CCOPY abc.txt
hello world
Welcome to C World!
#
```

图 10-8　例 10.12 运行结果——完成键盘输入复制功能

2．完成文件复制功能

```
#include "stdio.h"
include "stdlib.h"
main(int argc,char *argv[])
{  FILE *fp1,*fp2;
   char ch;
   if(argc!=3)
   {  printf("Input arguments error!\n");
      exit(0);
   }
   if((fp1=fopen(argv[1],"r"))==NULL)
   {  printf("The file %s can\'t open!\n",argv[1]);
      exit(0);
   }
   fp2=fopen(argv[2],"w");
   while((ch=fgetc(fp1))!=EOF)
      fputc(ch,fp2);
   fclose(fp1);
   fclose(fp2);
}
```

假设该程序以文件名“CFCOPY.C”存盘，经过编译连接生成可执行文件“CFCOPY.EXE”。“#”号之前的内容被存入文本文件“abc.txt”中。
运行方法参看图 10-9，结果参看 aabbcc.txt。

```
C:\Users\Administrator>CFCOPY abc.txt aabbcc.txt
```

图 10-9　例 10.12 运行结果——完成文件复制功能

3．完成两个文件的连接功能

```
#include "stdio.h"
```

```
include "stdlib.h"
main(int argc,char *argv[])
{  FILE *fp1,*fp2;
   char ch;
   if(argc!=3)
   {  printf("Input arguments error!\n");
      exit(0);
   }
   if((fp1=fopen(argv[1],"a"))==NULL)
   {  printf("The file %s can\'t open!\n",argv[1]);
      exit(0);
   }
   if((fp2=fopen(argv[2],"r"))==NULL)
   {  printf("The file %s can\'t open!\n",argv[2]);
      exit(0);
   }
   while((ch=fgetc(fp2))!=EOF)
      fputc(ch,fp1);
   fclose(fp1);
   fclose(fp2);
}
```

假设该程序以文件名“CLCOPY.C”存盘，经过编译连接生成可执行文件“CLCOPY.EXE”。运行方法见图 10-10，结果参看 abc.txt。

图 10-10　例 10.12 运行结果——完成两个文件的连接功能

在文件 abc.txt 中，可以看到其内容为原 abc.txt 中的内容与 aabbcc.txt 中的内容之和，并且原 abc.txt 中的内容在前，aabbcc.txt 中内容在后。

本 章 小 结

对于大多数 C 程序来说，向文件写入和从文件读出都是必需的。为实现这一目的，C 语言提供了可移植性更好的标准高级 I/O 服务。标准 I/O 服务将自动创建输入/输出缓冲区，以加快数据传输的速度。

fopen()函数为标准 I/O 打开一个文件，并创建一个用与存放有关文件和缓冲区信息的数据结构。fopen()函数返回指向这一数据结构的指针，其他函数可以用这个指针来指定要处理的文件。feof()函数和 ferror()函数向 I/O 报告操作失败的原因。fread()函数会将输入数据看成要被放置到指定存储位置的二进制值。如果使用 fscanf()函数，则将每个字节看成一个字符编码。fscanf()函数可以将字符编码翻译成格式说明符指定的其他类型。例如，%f 说明符会将输入翻译成一个浮点值，%d 说明符会将该输入翻译成一个整数值，而%s 说明符则会将这个字符输入保存为一个字符串。fgets()函数将输入保存为字符编码，把它作为独立的字符保存在字符变量中，或者作为字符串保存在字符数组中。与之类似，fwrite()函数将二进制数据直接放到输出流中，而其他输出函数将非字符数据转换成字符表示后再将其放到输出流中。

fgets()函数、fscanf()函数和 fread()函数一般从文件头开始顺序读取文件，而 fseek()函数和 ftell()函数允许程序移动到文件中任意位置进行随机存取。

习 题 10

一、填空题

1．文件是指_____。

2．根据数据的组织形式，C 语言中将文件分为_____和_____两种类型。

3．现要求将上题中打开的文件关闭，写出语句：_____。

4．若要用 fopen()函数打开一个新的二进制文件，该文件要既能读也能写，则打开文件方式字符串应该是_____。

5．若执行 fopen()函数时发生错误，则函数的返回值是_____。

6．若用 fopen()函数打开一个新的二进制文件，该文件要既能读也能写，则打开文件方式字符串应是_____。

7．fscanf()函数的正确调用形式是_____。

8．fgetc()函数的作用是从指定文件读入一个字符，该文件的打开方式必须是_____。

9．在执行 fopen()函数时，ferror()函数的初值是_____。

10．_____用来测试文件操作的一些函数是否出现错误。

二、选择题

1．以下叙述中错误的是（　　）。

A．在 C 语言中，对二进制文件的访问速度比文本文件快

B．在 C 语言中，随机文件以二进制代码形式存储数据

C．语句 FILEfp;定义了一个名为 fp 的文件指针

D．C 语言中的文本文件以 ASCII 码形式存储数据

2．若有以下程序：

```
#include<stdio.h>
int main(void)
{  FILE *fp;int i,k,n;
   fp=fopen("data.dat","w+");
   for(i=1;i<6;i++)
   {  fprintf(fp,"%d",i);
      if(i%3==0)
      fprintf(fp,"\n");
   }
   rewind(fp);
   fscanf(fp,"%d%d",&k,&n);
   printf("%d%d\n",k,n);
   fclose(fp);
   return 0;
}
```

程序运行后的输出结果是（　）。

A．00　　B．12345　　C．14　　D．12

3．以下与函数 fseek(fp,0L,SEEK_SET)有相同作用的是（　）。

A．feof(fp)　　B．ftell(fp)　　C．fgetc(fp)　　D．rewind(fp)

4．有如下程序：

```
#include<stdio.h>
int main(void)
{  FILE *fp1;
   fp1=fopen("f1.txt","w");
   fprintf(fp1,"abc");
   fclose(fp1);
   return 0;
}
```

若文本文件 f1.txt 中原有内容为：good，则运行以上程序后文件 f1.txt 中的内容为（　）。

A．goodabc　　B．abcd　　C．abc　　D．abcgood

5．以下程序的功能是（　）。

```
int main(void)
{  FILE *fp;
   char str[]="Beijing2008";
   fp=fopen("file2","w");
   fputs(str,fp);
   fclose(fp);
   return 0;
}
```

A．在屏幕上显示“Beijing2008”

B．把“Beijing2008”存入 file2 文件中

C．在打印机上打印出“Beijing2008”

D．以上都不对

6．有以下程序（提示：程序中 fseek(fp,-2L*sizeof(int),SEEK_END);语句的作用是使位置指针从文件尾向前移 2*sizeof(int)字节）。

```
int main(void)
{  FILE *fp;
   int i,a[4]={1,2,3,4},b;
   fp=fopen("data.dat","wb");
   for(i=0;i<4;i++)
      fwrite(&a[i],sizeof(int),1,fp);
   fclose(fp);
   fp=fopen("data.dat","rb");
   fseek(fp,-2L*sizeof(int),SEEK_END);
   fread(&b,sizeof(int),1,fp);      /*读取 sizeof(int)字节数据到变量 b 中*/
   fclose(fp);
   printf("%d\n",b);
   return 0;
}
```

则执行后输出结果是（　）。

A．2　　B．1　　C．4　　D．3

7．若 fp 已正确定义并指向某个文件，当未遇到该文件结束标志时函数 feof(fp)的值为（　）。

A．0　　B．1　　C．−1　　D．一个非 0 值

8．下列关于 C 语言数据文件的叙述中正确的是（　）。

A．文件由 ASCII 码字符序列组成，C 语言只能读/写文本文件

B．文件由二进制数据序列组成，C 语言只能读写二进制文件

C．文件由记录序列组成，可按数据的存放形式分为二进制文件和文本文件

D．文件由数据流形式组成，可按数据的存放形式分为二进制文件和文本文件

9．以下叙述中不正确的是（　）。

A．C 语言中的文本文件以 ASCII 码形式存储数据

B．在 C 语言中，对二进制文件的访问速度比文本文件快

C．在 C 语言中，随机读/写方式不适用于文本文件

D．在 C 语言中，顺序读/写方式不适用于二进制文件

10．以下程序企图把从终端输入的字符输出到名为 abc.txt 的文件中，直到从终端读入字符#号时结束输入和输出操作，但程序有错。

```
#include<stdio.h>
int main(void)
{  FILE*fout;char ch;
   fout=fopen("abc.txt",'w');
   ch=fgetc(stdin);
   while(ch!='#')
   {  fputc(ch,fout);
   ch=fgetc(stdin);
   }
   fclose(fout);
   return 0;
}
```

则出错的原因是（　）。

A．函数 fopen 调用形式错误　　B．输入文件没有关闭

C．函数 fgetc 调用形式错误　　D．文件指针 stdin 没有定义

11．有以下程序：

```
#include<stdio.h>
int main(void)
{  FILE*fp;int i=20,j=30,k,n;
   fp=fopen("d1.dat","w");
   fprintf(fp,"%d\n",i);
   fprintf(fp,"%d\n",j);
   fclose(fp);
   fp=fopen("d1.dat","r");
   fp=fscanf(fp,"%d%d",&k,&n);
   printf("%d%d\n",k,n);
   fclose(fp);
```

```
    return 0;
}
```

则程序运行后的输出结果是（　）。

A．2030　　B．2050　　C．3050　　D．3020

12．以下叙述中错误的是（　）。

A．二进制文件打开后可以先读文件的末尾，而顺序文件不可以

B．在程序结束时，应当用 fclose 函数关闭已打开的文件

C．在利用 fread 函数从二进制文件中读数据时，可以用数组名给数组中所有元素读入数据

D．不可以用 FILE 定义指向二进制文件的文件指针

13．若要打开 A 盘上 user 子目录下名为 abc.txt 的文本文件进行读、写操作，下面符合此要求的函数调用是（　）。

A．fopen("A:\user\abc.txt","r")　　B．fopen("A:\\user\\abc.txt","r+")

C．fopen("A:\user\abc.txt","rb")　　D．fopen("A:\\user\\abc.txt","w")

14．下面的程序执行后，文件 test.t 中的内容是（　）。

```
#include<stdio.h>
void fun(char*fname ,char *st)
{  FILE *myf;
   int i;
   myf=fopen(fname,"w");
   for(i=0;i<strlen(st);i++)
   fputc(st[i],myf);
   fclose(myf);
}
int main(void)
{  fun("test","new world");
   fun("test","hello")
   return 0;
}
```

A．hello，　　B．newworld hello，　　C．newworld　　D．hello,rld

15．若 fp 是指向某文件的指针，且已读到文件末尾，则库函数 feof(fp)的返回值是（　）。

A．EOF　　B．–1　　C．非零值　　D．NULL

16．在 C 程序中，可把整型数以二进制形式存放到文件中的函数是（　）。

A．fprintf　　B．fread　　C．fwrite　　D．fputc

17．标准函数 fgets(s,n,f)的功能是（　）。

A．从文件 f 中读取长度为 n 的字符串存入指针 s 所指的内存

B．从文件 f 中读取长度不超过 n–1 的字符串存入指针 s 所指的内存

C．从文件 f 中读取 n 个字符串存入指针 s 所指的内存

D．从文件 f 中读取长度为 n–1 的字符串存入指针 s 所指的内存

18．利用 fseek()函数可实现的操作是（　）。

A．fseek(文件类型指针,起始点,位移量);

B．fseek(fp,位移量,起始点);

C．fseek(位移量,起始点,fp);

D．fseek(起始点,位移量,文件类型指针);

19．在执行 fopen()函数时，ferror()函数的初值是（　　）。

A．TURE　　B．–1　　C．1　　D．0

20．fread(buf,64,2,fp)的功能是（　　）。

A．从 fp 所指向的文件中，读出整数 64，并存放在 buf 中

B．从 fp 所指向的文件中，读出整数 64 和 2，并存放在 buf 中

C．从 fp 所指向的文件中，读出 64 个字节的字符，读两次，并存放 buf 地址中

D．从 fp 所指向的文件中，读出 64 个字节的字符，并存放在 buf 中

三、程序填空题

1．已有文本文件 test.txt，其中的内容为：Hello,everyone!。在以下程序中，文件 test.txt 已正确为“读”而打开，由文件指针 fr 指向该文件，则程序的输出结果是_____。

```
#include <stdio.h>
int main(void)
{  FILE *fr;char str[40];
   fgets(str,5,fr);
   printf("%s\n",str);
   fclose(fr);
   return 0;
}
```

2．若 fp 已正确定义为一个文件指针，d1.dat 为二进制文件，请填空，以便为“读”而打开此文件：fp=fopen(_____);。

3．以下程序用来统计文件中字符个数。请填空。

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{  FILE *fp;long num=0L;
   if((fp=fopen("fname.dat","r"))= =NULL)
   {  printf("Openerror\n");
      exit(0);
   }
   while(_____)
   fgetc(fp);num++;
   printf("num=%1d\n",num-1);
   fclose(fp);
   return 0;
}
```

4．以下程序段打开文件后，先利用 fseek 函数将文件位置指针定位在文件末尾，然后调用 ftell 函数返回当前文件位置指针的具体位置，从而确定文件长度。请填空。

```
int main(void)
{  FILE *myf;
   long f1;
   myf=_____("test.txt","rb");
```

```
    fseek(myf,0,SEEK_END);
    f1=ftell(myf);
    fclose(myf);
    printf("%d\n",f1);
    return 0;
}
```

5．下面程序把从终端读入的文本（用@作为文本结束标志）输出到一个名为 bi.dat 的新文件中。请填空。

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{  FILE *fp;
   char ch;
   if((fp=fopen(_________))= =NULL)
      exit(0);
   while((ch=getchar())!='@')
      fputc(ch,fp);
   fclose(fp);
   return 0;
}
```

6．在以下程序中，用户由键盘输入一个文件名，然后输入一串字符（用#结束输入）存放到此文件文件中形成文本文件，并将字符的个数写到文件尾部。请填空。

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{  FILE *fp;
   char ch,fname[32];
   int count=0;
   printf("Input the filename: ");
   scanf("%s",fname);
   if((fp=fopen(_________,"w+"))= =NULL)
   {  printf("Can't open file: %s\n",fname);
      exit(0);
   }
   printf("Enterdata: \n");
   while((ch=getchar())!="#"){
      fputc(ch, fp);
      count++;
   }
   fprintf(_________,"\n%d\n",count);
   fclose(fp);
   return 0;
}
```

7．下面程序把从终端读入的 10 个整数以二进制方式写到一个名为 bi.dat 的新文件中，请填空。

```
#include <stdio.h>
#include <stdlib.h>
FILE*fp;
int main(void)
{  int i,j;
   if((fp=fopen(________,"wb"))= =NULL)
      exit(0);
   for(i=0;i<10;i++)
   {  scanf("%d",&j);
      fwrite(&j,sizeof(int),1,________);
   }
   fclose(fp);
   return 0;
}
```

8. 以下程序的功能是：从键盘上输入一个字符串，把该字符串中的小写字母转换为大写字母，输出到文件 test.txt 中，然后从该文件读出字符串并显示出来。请填空。

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{  FILE *fp;
   char str[100];
   int i=0;
   if((fp=fopen("text.txt",________))==NULL)
   {  printf("can'topenthisfile.\n");
      exit(0);
   }
   printf("inputastring:\n");gets(str);
   while(str[i])
   {  if(str[i]>='a'&&str[i]<='z')
         str[i]=________;
      fputc(str[i],fp);
      i++;
   }
   fclose(fp);
   fp=fopen("test.txt",________);
   fgets(str,100,fp);
   printf("%s\n",str);
   fclose(fp);
   return 0;
}
```

9. 以下 C 语言程序将磁盘中的一个文件复制到另一个文件中，两个文件名在命令行中给出。

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc,char *argv)
{  FILE *f1,*f2;
```

```
    char ch;
    if(argc<________)
    {  printf("Parametersmissing!\n");
       exit(0);}
    if(((f1=fopen(argv[1],"r"))==NULL)||((f2=fopen(argv[2],"w"))==NULL))
    {  printf("Cannotopenfile!\n");
       exit(0);}
    while(________)
       fputc(fgetc(f1),f2);
    fclose(f1);
    fclose(f2);
    return 0;
}
```

四、程序设计题

1．将程序中的一组学生记录以文件名 stufile.txt 形式存盘，以便永久保留下来，其中每个学生记录包含学号、姓名、计算机、英语、总分等项。

2．将上述文件 stufile.txt 中的记录按总分降序排列后存入到文件 stusort.txt 中。

3．将上述文件 stufile.txt 中的记录按总分降序排列，总分相同的记录按学号升序排列后存入到文件 stusort.txt 中。

4．在磁盘文件上存有 10 个学生的数据。要求将第 1、3、5、7、9 个学生数据输入计算机，并在屏幕上显示出来。

附录 A　常用字符与 ASCII 代码对照表

ASCII 值	控制字符	字符	ASCII 值	字符	ASCII 值	字符	ASCII 值	字符	ASCII 值	字符	ASCII 值	字符	ASCII 值	字符	ASCII 值	字符
000	null	NUL	032	(space)	064	@	096	’	128	Ç	160	á	192	└	224	α
001	☺	SOH	033	!	065	A	097	a	129	ü	161	í	193	┴	225	β
002	☻	STX	034	"	066	B	098	b	130	é	162	ó	194	┬	226	Γ
003	♥	ETX	035	#	067	C	099	c	131	â	163	ú	195	├	227	π
004	♦	EOT	036	$	068	D	100	d	132	ã	164	ñ	196	—	228	Σ
005	♣	END	037	%	069	E	101	e	133	à	165	Ñ	197	†	229	σ
006	♠	ACK	038	&	070	F	102	f	134	å	166	ª	198	╞	230	μ
007	beep	BEL	039	'	071	G	103	g	135	ç	167	º	199	╟	231	τ
008	backspace	BS	040	(	072	H	104	h	136	ê	168	¿	200	╚	232	Φ
009	tab	HT	041	)	073	I	105	i	137	ë	169	⌐	201	╔	233	θ
010	换行	LF	042	*	074	J	106	j	138	è	170	¬	202	╩	234	Ω
011	♂	VT	043	+	075	K	107	k	139	ï	171	½	203	╦	235	δ
012	♀	FF	044	,	076	L	108	l	140	î	172	¼	204	╠	236	∞
013	回车	CR	045	-	077	M	109	m	141	ì	173	¡	205	═	237	ø
014	♫	SO	046	.	078	N	110	n	142	Ä	174	«	206	╬	238	ϵ
015	☼	SI	047	/	079	O	111	o	143	Å	175	»	207	╧	239	∩
016	►	DLE	048	0	080	P	112	p	144	É	176	░	208	╨	240	≡
017	◄	DC1	049	1	081	Q	113	q	145	æ	177	▒	209	╤	241	±
018	↕	DC2	050	2	082	R	114	r	146	Æ	178	▓	210	╥	242	≥
019	‼	DC3	051	3	083	S	115	s	147	ô	179	│	211	╙	243	≤
020	¶	DC4	052	4	084	T	116	t	148	ö	180	┤	212	╘	244	⌠
021	§	NAK	053	5	085	U	117	u	149	ò	181	╡	213	╒	245	⌡
022	▬	SYN	054	6	086	V	118	v	150	û	182	╢	214	╓	246	÷
023	↨	ETB	055	7	087	W	119	w	151	ù	183	╖	215	╫	247	≈
024	↑	CAN	056	8	088	X	120	x	152	ÿ	184	╕	216	╪	248	°
025	↓	EM	057	9	089	Y	121	y	153	ö	185	╣	217	┘	249	∙
026	→	SUB	058	:	090	Z	122	z	154	Ü	186	║	218	┌	250	·
027	←	ESC	059	;	091	[	123	{	155	¢	187	╗	219	█	251	√
028	└	FS	060	<	092	\	124	¦	156	£	188	╝	220	▄	252	ⁿ
029	↔	GS	061	=	093	]	125	}	157	¥	189	╜	221	▌	253	²
030	▲	RS	062	>	094	^	126	~	158	P_t	190	╛	222	▐	254	■
031	▼	US	063	?	095	_	127	⌂	159	*f*	191	┐	223	▀	255	

注：第 0～32 号及第 127 号（共 34 个）是控制字符或通信专用字符，如控制符：LF（换行）、CR（回车）、FF（换页）、DEL（删除）、BEL（振铃）等；通信专用字符：SOH（文头）、EOT（文尾）、ACK（确认）等；第 33～126 号（共 94 个）是字符，其中第 48～57 号为 0～9 十个阿拉伯数字；65～90 号为 26 个大写英文字母，97～122 号为 26 个小写英文字母，其余为一些标点符号、运算符号等。第 128～255 号为扩展字符（不常用）。

附录 B　C 语言中的关键字

C 语言的关键字共有 32 个，根据关键字的作用，可分为数据类型关键字、控制语句关键字、存储类型关键字和其他关键字四类。

1．数据类型关键字（12 个）

（1）char：声明字符型变量或函数。
（2）double：声明双精度变量或函数。
（3）enum：声明枚举类型。
（4）float：声明浮点型变量或函数。
（5）int：声明整型变量或函数。
（6）long：声明长整型变量或函数。
（7）short：声明短整型变量或函数。
（8）signed：声明有符号类型变量或函数。
（9）struct：声明结构体变量或函数。
（10）union：声明共用体（联合）数据类型。
（11）unsigned：声明无符号类型变量或函数。
（12）void：声明函数无返回值或无参数，声明无类型指针（基本上就这三个作用）。

2．控制语句关键字（12 个）

1）循环语句
（1）for：一种循环语句（可意会不可言传）。
（2）do：循环语句的循环体。
（3）while：循环语句的循环条件。
（4）break：跳出当前循环。
（5）continue：结束当前循环，开始下一轮循环。

2）条件语句
（1）if：条件语句。
（2）else：条件语句否定分支（与 if 连用）。
（3）goto：无条件跳转语句。

3）开关语句
（1）switch：用于开关语句。
（2）case：开关语句分支。
（3）default：开关语句中的“其他”分支。

4）返回语句
return：子程序返回语句（可以带参数，也可不带参数）。

3．存储类型关键字（4 个）

（1）auto：声明自动变量，一般不使用。
（2）extern：声明变量是在其他文件上声明（也可以看成引用变量）。
（3）register：声明寄存器变量。
（4）static：声明静态变量。

4．其他关键字（4 个）

（1）const：声明只读变量。
（2）sizeof：计算数据类型长度。
（3）typedef：用以给数据类型取别名（当然，还有其他作用）。
（4）volatile：说明变量在程序执行中可被隐含地改变。

附录 C　C 语言库函数

库函数并不是 C 语言的一部分。它是由人们根据需要编制并提供用户使用的。每一种 C 编译系统都提供了一批库函数，不同的编译系统所提供的库函数的数目和函数名，以及函数功能是不完全相同的。ANSI C 标准提出了一批建议提供的标准库函数。它包括了目前多数 C 编译系统所提供的库函数，但也有一些是某些 C 编译系统未曾实现的。考虑到通用性，本书列出 ANSI C 标准建议提供的、常用的部分库函数。对多数 C 编译系统，可以使用这些函数的绝大部分。由于 C 库函数的种类和数目很多（例如，还有屏幕和图形函数、时间日期函数、与系统有关的函数等，每一类函数又包括各种功能的函数），本附录不能全部介绍，只从教学需要的角度列出最基本的。读者在编制 C 程序时可能要用到更多的函数，请查阅所用系统的手册。

1．数学函数

使用数学函数时，应该在源文件中使用：#include "math.h"。数学函数见表 C-1。

表 C-1　数学函数

函数名	函数类型和形参类型	功能	返回值	说明
acos	double acos(x) double x;	计算反余弦 arccos(x)的值	计算结果	应在–1～1 范围内
asin	double asin(x) double x;	计算反正弦 arcsin(x)的值	计算结果	应在–1～1 范围内
atan	double atan(x) double x;	计算反正切 arctan(x)的值	计算结果	
atan2	double atan2(x,y) double x,y;	计算 arctan(y/x)的值	计算结果	
cos	double cos(x) double x;	计算余弦 cos(x)的值	计算结果	x 的单位为弧度
cosh	double cosh(x) double x;	计算 x 的双曲余弦 cosh(x)的值	计算结果	
exp	double exp(x) double x;	计算指数 e^x 的值	计算结果	
fabs	double fabs(x) double x;	计算 x 的绝对值	计算结果	
floor	double floor(x) double x;	求出不大于 x 的最大整数	该整数的双精度实数	
fmod	double fmod(x,y) double x,y;	求整除 x/y 的余数	返回余数的双精度实数	
frexp	double frexp(val,eptr) double val; int *eptr;	把双精度数 val 分解为数字部分（尾数）x 和以 2 为底的指数 n，存放在 eptr 指向的变量中	返回数字部分	
log	double log(x) double x;	求自然对数 ln(x)的值	计算结果	
log10	double log10(x) double x;	求以 10 为底的对数 lg(x)的值	计算结果	
modf	double mode(val,iptr) double val; double *iptr;	把双精度数 val 分解为整数部分和小数部分，把整数存放到 iptr 指向的单元	小数部分	
pow	double pow(x,y) double x,y;	求 x^y 的值	计算结果	
sin	double sin(x) double x;	求正弦函数 sin(x)的值	计算结果	
sint	ouble sinh(x) double x;	计算 x 的双曲正弦函数 sinh(x)的值	计算结果	
sqrt	double sqrt(x) double x;	计算 x 的平方根	计算结果	

续表

函数名	函数类型和形参类型	功能	返回值	说明
tan	double tan(x) double x;	计算正切函数 tan(x)的值	计算结果	
tanh	double tanh(x) double x;	计算 x 的双曲正切函数 tanh(x)的值	计算结果	

2．字符函数和字符串函数

ANSI C 标准要求在使用字符串函数时要包含头文件“string.h”，在使用字符函数时要包含头文件“ctype.h”。有的 C 编译不遵循 ANSI C 标准的规定，而用其他名称的头文件。请使用时查有关手册。字符函数和字符串函数见表 C-2。

表 C-2　字符函数和字符串函数

函数名	函数和行参类型	功能	返回值	包含文件
isalnum	int isalnum(ch) int ch;	检查 ch 是否是字母(alpha)或数字(numeric)	是字母或数字返回 1；否则返回 0	ctype.h
isalpha	int isalpha (ch) int ch;	检查 ch 是否是字母字符	是，返回 1； 不是，返回 0	ctype.h
iscntrl	int iscntrl (ch) int ch;	检查 ch 是否是控制字符（其 ASCII 码在 0x7f 或 0x00～0x1F 之间）	是，返回 1； 不是，返回 0 （不包括空格）	ctype.h
isdigit	int isdigit (ch) int ch;	检查 ch 是否是数字（0～9）	是，返回 1； 不是，返回 0	ctype.h
isgraph	int isgraph (ch) int ch;	检查 ch 是否是可打印字符（其 ASCII 码在 0x21～0x7E 之间）	是，返回 1； 不是，返回 0	ctype.h
islower	int islower (ch) int ch;	检查 ch 是否是小写字母（a～z）	是，返回 1； 不是，返回 0	ctype.h
isprint	is isprint (ch) int ch;	检查 ch 是否是可打印字符，其 ASCII 码在 0x20～0x7E 之间	是，返回 1； 不是，返回 0	ctype.h
ispunct	int ispunct (ch) int ch;	检查 ch 是否是标点字符（不包括空格），即除字母、数字和空格以外的所有可打印字符	是，返回 1； 不是，返回 0	ctype.h
isspace	int isspace (ch) int ch;	检查 ch 是否是空格、跳格符（制表符）或换行符	是，返回 1； 不是，返回 0	ctype.h
isupper	int isupper (ch) int ch;	检查 ch 是否是大写字母（A～Z）	是，返回 1； 不是，返回 0	ctype.h
isxdigit	int isxdigit (ch) int ch;	检查 ch 是否是十六进制数(0～9，A～F,a～f)	是，返回 1； 不是，返回 0	ctype.h
strcat	char*strcat(str1,str2) char *str,*str2;	把字符串 str2 接到 str1 后面，str1 最后面的'\0'被取消	str1	string.h
strchr	char *strchr (str,ch) char *str; int ch;	找出 str 指向的字符串中第一次出现字符 ch 的位置	返回指向该位置的指针，如找不到，则返回空指针	string.h
strcmp	int strcmp(str1,str2) char　*str,*str2;	比较两个字符串 str1、str2	str1<str2，返回负数 str1=str2，返回 0 str1>str2，返回正数	string.h
strcpy	char *strcpy(str1,str2) char *str1,*str2;	把 str2 指向的字符串复制到 str1 中去	返回 str1	string.h
strlen	unsigned int strlen (str) char *str;	统计字符串 str 中字符的个数（不包括终止符'\0'）	返回字符个数	string.h
strstr	char*strstr(str1,str2) char *str1,*str2;	找出 str2 字符串中在 str1 字符串中第一次出现的位置（不包括 str2 的串结束符）	返回该位置的指针。如找不到，则返回空指针	string.h
tolower	int　tolower (ch) int　ch;	将 ch 字符转换为小写字母	返回 ch 所代表的字符的小写字母	ctype.h
toupper	int　toupper (ch) int　ch;	将 ch 字符转换为大写字母	与 ch 字符相应的大写字母	ctype.h

3．输入/输出函数

使用如表 C-3 所示的输入/输出函数，应该把 stdio.h 头文件包含到源程序文件中。

表 C-3　输入/输出函数

函数名	函数和形参类型	功能	返回值	说明
clearerr	void clearerr (fp) FILE *fp;	清除文件指针错误指示器	无	
fclose	int fclose (fp) FILE *fp;	关闭所指的文件，释放文件缓冲区	有错则返回非 0，否则返回 0	
feof	int feof (fp) FILE *fp;	检查文件是否结束	遇文件结束符返回非零值，否则返回 0	
fgetc	int fget (fp) FILE *fp;	从 fp 所指定的文件中取得下一个字符	返回所得到的字符。若读入出错，返回 EOF	
fgets	char *fgets(buf,n,fp) char *buf; int n; FILE *fp;	从 fp 指向的文件读取一个长度为(n–1)的字符串，存入起始地址为 buf 的空间	返回地址 buf，若遇文件结束或出错，返回 NULL	
fopen	FILE *fopen (filename,mode) char *filename, *mode;	以 mode 指定的方式打开名为 filename 的文件	成功，返回一个文件指针（文件信息区的起始地址），否则返回 0	
fprintf	int fprintf (fp, format,args,…) FILE *fp; char *format;	把 args 的值以 format 指定的格式输出到 fp 所指定的文件中	实际输出的字符数	
fputc	int fputc (ch,fp) char ch; FILE *fp;	将字符 ch 输出到 fp 指向的文件中	成功，则返回该字符；否则返回 EOF	
fputs	int fputs (str,fp) char *str; FILE *fp;	将 str 指向的字符串输出到 fp 所指定的文件	返回 0，若出错，则返回非 0	
fread	int fread (pt,size,n,fp) char *pt; unsigned size; unsigned n;FILE *fp;	从 fp 所指定的文件中读取长度为 size 的 n 个数据项，存到 pt 所指向的内存区	返回所读的数据项的个数，如遇文件结束或出错，则返回 0	
fscanf	int fscanf (fp,format,args,…) FILE *fp; char *format;	从 fp 指定的文件中按 format 给定的格式将输入数据送到 args 所指向的内存单元(args 是指针)	输入的数据个数	
fseek	int fseek (fp,offset,base) FILE *fp; long offset; int base;	将 fp 所指向的文件的位置指针移到以 base 所指出的位置为基准、以 offset 为位移量的位置	返回当前位置，否则返回–1	
ftell	long ftell (fp) FILE *fp;	返回 fp 所指向的文件中的读/写位置	返回 fp 所指向的文件中的读/写位置	
fwrite	int fwrite (ptr,size,n,fp) char *ptr; unsigned size; unsigned n; FILE *fp;	把 ptr 所指向的 n*size 个字节输出到 fp 所指向的文件中	写到 fp 文件中的数据项的个数	
getc	int getc (fp) FILE *fp;	从 fp 所指向的文件中读入一个字符	返回所读的字符，若文件结束或出错，则返回 EOF	
getchar	int getchar(void)	从标准输入设备读取下一个字符	所读字符。若文件结束或出错，则返回–1	
getw	int getw (fp) FILE *fp;	从 fp 所指向的文件读取下一个字（整数）	输入的整数。若文件结束或出错，则返回–1	非 ANSI 标准
printf	int printf (format,args,…) char *format;	将输出表列 args 的值输出到标准输出设备	输出字符的个数，若出错，返回负数	Format 可以是一个字符串，或字符数组的起始地址
putc	int putc (ch, fp) int ch; FILE *fp;	把一个字符 ch 输出到 fp 所指的文件中	输出的字符 ch，若出错，返回 EOF	

续表

函数名	函数和形参类型	功能	返回值	说明
putchar	int putchar (ch) char ch;	把字符 ch 输出到标准输出设备	输出的字符 ch。若出错，返回 EOF	
puts	int puts (str) char *str;	把 str 指向的字符串输出到标准输出设备，将'\0'转换为回车换行	返回换行符。若失败，返回 EOF	
putw	int putw (w,fp) int w; FILE *fp;	将一个整数 w（即一个字）写到 fp 指向的文件中	返回输出的整数，若出错，返回 EOF	非 ANSI 标准
rename	int rename (oldname, newname) char *oldname,*newname;	把由 oldname 所指向的文件名改为由 newname 所指向的文件名	成功返回 0， 出错返回–1	
rewind	void rewind (fp) FILE *fp;	将 fp 指向的文件中的位置指针置于文件开头位置，并清除文件结束标志和错误标志	无	
scanf	int scanf (format,args, …) char *format;	从标准输入设备按 format 指向的格式字符串规定的格式，输入数据给 args 所指向的单元	读入并赋给 args 的数据个数。遇文件结束返回 EOF，出错返回 0	args 为指针

4．动态存储分配函数

ANSI 标准建议设 4 个有关的动态存储分配的函数，即 calloc()、malloc()、free()、realloc()。实际上，许多 C 编译系统实现时，往往增加了一些其他函数。ANSI 标准建议在"stdlib.h"头文件中包含有关的信息，但许多 C 编译要求用"malloc.h"而不是"stdlib.h"。读者在使用时应查阅有关手册。

ANSI 标准要求动态分配系统返回 void 指针。void 指针具有一般性，它们可以指向任何类型的数据。但目前绝大多数 C 编译所提供的这类函数都返回 char 指针。无论是以上两种情况的哪一种，都需要用强制类型转换的方法把 char 指针转换成所需的类型。动态存储分配函数见表 C-4。

表 C-4　动态存储分配函数

函数名	函数和形参类型	功能	返回值
calloc	void (或 char) *calloc(n,size) unsigned n;unsigned size;	分配 n 个数据项的内存连续空间，每个数据项的大小为 size	所分配内存单元的起始地址。如不成功，返回 0
free	void free(p) void (或 char) *p;	释放 p 所指的内存区	无
malloc	void (或 char) *malloc(size) unsigned size;	分配 size 字节的存储区	所分配的内存单元的起始地址。如内存不够，返回 0
realloc	void (或 char) *realloc(p,size) void(或 char) *p; unsigned size;	将 p 所指向的已分配内存区的大小改为 size。size 可以比原来分配的空间大或小	返回指向新内存区的指针

5．图形函数

在使用 C 语言作图时，应包括头文件：graphics.h。图形函数见表 C-5。

表 C-5　图形函数

函数名	函数和形参类型	功能	返回值
initgraph	void far initgraph(driver,mode,path) int far *driver,mode;char far path;	按指定图形模式初始化图形系统	无
detectgraph	void far detectgraph(graphdriver,graphmode) int far *graphdriver,*graphmode;	检测所用的显示适配器类型	无
cleardevice	void far cleardevice(void)	清屏幕	无
clearviewport	void far clearviewport(void)	清除图视口区域	无
closegraph	void far closegraph(void)	关闭图形方式，返到文本方式	无
restorecrtmode	void far restorecrtmode(void)	恢复工作模式，同时清屏	无

续表

函数名	函数和形参类型	功能	返回值
putpixel	void far putpixel(x,y,color) int x,y,color;	在(x,y)坐标处用 color 颜色画点	无
getpixel	int far getpixel(x,y) int x,y;	得到(x,y)坐标处的颜色值	
moveto	void far moveto(x,y) int x,y;	移动画笔到指定的(x,y)坐标处	无
moverel	void far moverel(dx,dy) int dx,dy;	dx,dy 为从当前位置相对移动的纵横位置量	无
getx	int far getx(void)	获取当前 x 坐标值	
gety	int far gety(void)	获取当前 y 坐标值	
line	void far line(x_1,y_1,x_2,y_2) int x_1,y_1,x_2,y_2;	从(x_1,y_1)到(x_2,y_2)画一直线	无
lineto	void far lineto(x,y) int x,y;	从当前位置到(x,y)处画一直线	无
linerel	void far linerel(dx,dy) int dx,dy;	从当前位置到 x,y 增量处画线	无
rectangle	void far rectangle(x_1,y_1,x_2,y_2) int x_1,y_1,x_2,y_2;	画一(x_1,y_1)为左上角，(x_2,y_2)为右下角的矩形框	无
bar	void bar(x_1,y_1,x_2,y_2) int x_1,y_1,x_2,y_2;	画一(x_1,y_1)为左上角，(x_2,y_2)为右下角的条状图	无
ellipse	void ellipse (x,y,stangle,endangle,xradius,yradius) int x,y, stangle,endangle; int xradius,yradius;	画一椭圆线	无
circle	void far circle(x,y,radius) int x,y,radius;	以 radius 半径(x,y)为圆心画圆	无
arc	void far arc (x,y,stangle,endangle,radius) int x,y,stangle,endangle,radius;	以 stangle 为起始角，endangle 为终止角画弧	无
pieslice	void far pieslice (x,y,stangle,endangle,radius) int x,y,stangle,endangle,radius;	同上，但画的是一个扇形弧	无
setcolor	void far setcolor(int color)	设置前景色	无
setbkcolor	void far setbkcolor(int color)	设置背景色	无
setpalette	void far setpalette (int index,int actual_color)	调色板颜色设置	无
setallpalette	void far setallpalette (struct palettetype far *palette)	改变调色板 16 种颜色	无
getpalette	void far getpalette (struct palettetype far *palette)	得到调色板的颜色数和赋予的颜色值	无
getpalettesize	void far getpalettesize(void)	得出调色板颜色数	无
setlinestyle	void far setlinestyle (linestyle,upattern,thickness) int linestyle,thickness; unsigned upattern;	设定线型	无
getlinesettings	void far getlinesettings (struct linesettingstype far *lineinfo)	得到当前有关线型	无
setfillstyle	void far setfillstyle(pattern,color) int pattern,color;	用颜色和图模对轮廓图填充	无
setfillpattern	void far setfillpattern (char *upattern,int color)	用户自定义可填充模式	无
fillsettings	void far fillsettings (struct fillsettingstype far *fillinfo)	得到当前填充模式和颜色	无
bar3d	void far bar3d (x_1,y_1,x_2,y_2,depth,topflag) int x_1,y_1,x_2,y_2,depth,topflag;	画三维立体直方图。topflag=0:无顶，否则有顶	无
sector	void far sector (x,y,stangle,endangle,xradius,yradius) int x,y,stangle,endangle,xradius,yradius;	画椭圆扇形，stangle:起始角，endangle:终止角	无
fillellipse	void far fillellipse (int x,int y,int xradius,int yradius)	画椭圆填充图	无
fillpoly	void far fillpoly (int numpoints,int far *palypoints)	画多边形填充图	无

续表

函数名	函数和形参类型	功能	返回值
floodfill	void far floodfill(x,y,border) int x,y,border;	淹没式填充，遇边界颜色停	无
getimage	void far getimage(x_1,y_1,x_2,y_2,bitmap) int x_1,y_1,x_2,y_2;void far *bitmap;	存屏幕图像到内存	无
imagesize	unsigned far imagesize (int x_1,int y_1,int x_2,int y_2)	测试矩形区图像所占字节数	
putimage	void far putimage (int x,int y,void far *bitmap,int op)	把 bitmap 中所存图像进行 op 操作后显示到(x,y)开始处	无
setactivepage	void far setactivepage(int pagenum)	设置激活页	无
setvisualpage	void far setvisualpage(int pagenum)	设置显示页	无
setviewport	void far setviewport (int x_1,int y_1,int x_2,int y_2,int clipflag)	设置图视口	无
clearviewport	void far clearviewport(void)	清除图视口	无
getviewsettings	void far getviewsettings(viewport) struct viewporttype far *viewport;	取当前图视口信息到 viewport	无
outtext	void far outtext(char far *textstring)	在当前位置输出文本	无
outtextxy	void far outtextxy(x,y,textstring) int x,y;char far *textstring;	在(x,y)位置输出文本	无
settextjustify	void far settextjustify(horiz,vert) int horiz,vert;	设置文本输出的对齐方式	无
settextstyle	void far settextstyle (int font,int direction,charsize)	设置输出的字型、大小和方向	无
sprinf	int sprintf(string,farmat[,argument,…]) char *string,*format;	把 argument 的值按 format 格式写入 string 中	

6．文本屏幕输出函数

使用文本方式可以开窗口、着色、写数据等。所有函数说明在头文件：conio.h 中。文本屏幕输出函数见表 C-6。

表 C-6　文本屏幕输出函数

函数名	函数和形参类型	功能	返回值
textmode	void textmode(int newmode)	设置文本显示方式	无
textcolor	void textcolor(int color)	设置文本前景颜色	无
textbackground	void textbackground(int color)	设置文本背景颜色	无
textattr	void textattr(int attr)	设置文本显示属性	无
highvideo	void highvideo(void)	设置高亮度显示字符	无
lowvideo	void lowvideo(void)	设置低亮度显示字符	无
window	void window(x_1,y_1,x_2,y_2) int x_1,y_1,x_2,y_2;	以(x_1,y_1)为左上角点、(x_2,y_2)为右下角开窗口	无
cprintf	int cprintf(char *format,…)	按 format 格式写数据	
cputs	int cputs(char *str)	输出字符串	
putch	int putch(int ch)	输出字符	
clrscr	void clrscr(void)	清屏幕	无
clreol	void clreol(void)	从当前位置清除到本行末尾	无
delline	void delline(void)	清除光标所在行文本	无
gotoxy	void gotoxy(int x,int y)	光标定位到(x,y)处	无
movetext	void movetext (x_1,y_1,x_2,y_2,x_3,y_3) int x_1,y_1,x_2,y_2,x_3,y_3;	将(x_1,y_1)到(x_2,y_2)的矩形区移到(x_3,y_3)处	无
gettext	void gettext(x_1,y_1,x_2,y_2,buffer) int x_1,y_1,x_2,y_2;void *buffer;	将矩形区域信息存入 buffer 开始的单元中	无
puttext	void puttext(x_1,y_1,x_2,y_2,buffer) int x_1,y_1,x_2,y_2;void *buffer;	将 buffer 开始的单元中信息存入矩形区域内	无
gettextinfo	void gettextinfo(f) struct text_info *f;	得到屏幕显示有关信息的函数	无
wherex	int wherex(void)	获得当前光标 x 坐标	
wherey	int wherey(void)	获得当前光标 y 坐标	

7．DOS、BIOS 调用函数

DOS、BIOS 调用函数见表 C-7。使用该函数应包含头文件：dos.h。

表 C-7　DOS、BIOS 调用函数

函数名	函数和形参类型	功能	返回值
int86	int int86(num,in,out) int num; union REGS *in,*out	执行有 num 规定的 8086 软中断程序	
int86x	int int86(num,in,out,s) int num; union REGS *in,*out struct SREGS *s	执行有 num 规定的 8086 软中断程序，调用前保存 DS 和 ES 的值，调用后恢复	
intdos	int intdos(in,out) union REGS *in,*out	执行 0x21 中断，调用一定的 DOS 功能	
intdosx	int intdosx(in,out,s) union REGS *in,*out struct SREGS *s	执行 0x21 中断，调用一定的 DOS 功能。调用前保存 DS 和 ES 的值，调用后恢复	

附录 D　Visual C++ 6.0 编译错误信息

1．fatal error C1010: unexpected end of file while looking for precompiled header directive。
寻找预编译头文件路径时遇到了不该遇到的文件尾（一般是没有＃include "stdafx.h"）。
2．fatal error C1083: Cannot open include file: 'R…….h': No such file or directory
不能打开包含文件“R…….h”：没有这样的文件或目录。
3．error C2011: 'C……': 'class' type redefinition
类“C……”重定义。
4．error C2018: unknown character '0xa3'
不认识的字符'0xa3'（一般是汉字或中文标点符号）。
5．error C2057: expected constant expression
希望是常量表达式（一般出现在 switch 语句的 case 分支中）。
6．error C2065: 'I……' : undeclared identifier
“I……”：未声明过的标识符。
7．error C2082: redefinition of formal parameter 'b……'
函数参数“b……”在函数体中重定义。
8．error C2143: syntax error: missing ':' before '{'
语法错误：“{”前缺少“;”。
9．error C2146: syntax error: missing ';' before identifier 'dc'
语法错误：在“dc”前丢了“;”。
10．error C2196: case value '69' already used
值 69 已经用过（一般出现在 switch 语句的 case 分支中）。
11．error C2509: 'OnTimer': member function not declared in 'CHelloView'
成员函数“OnTimer”没有在“CHelloView”中声明。
12．error C2511: 'reset': overloaded member function 'void (int)' not found in 'B'
重载的函数“void reset(int)”在类“B”中找不到。
13．error C2555: 'B::f1': overriding virtual function differs from 'A::f1' only by return type or calling convention
类 B 对类 A 中同名函数 f1 的重载仅根据返回值或调用约定上的区别。
14．error C2660: 'SetTimer': function does not take 2 parameters
“SetTimer”函数不传递 2 个参数。
15．warning C4035: 'f……': no return value
“f……”的 return 语句没有返回值。
16．warning C4553: '= =': operator has no effect; did you intend '='?
没有效果的运算符“= =”；是否改为“=”？
17．warning C4700: local variable 'bReset' used without having been initialized
局部变量“bReset”没有初始化就使用。

18．error C4716: 'CMyApp::InitInstance': must return a value

“CMyApp::InitInstance”函数必须返回一个值。

19．LINK: fatal error LNK1168: cannot open Debug/P1.exe for writing

连接错误：不能打开 P1.exe 文件，以改写内容（一般是 P1.Exe 还在运行，未关闭）。

20．error LNK2001: unresolved external symbol "public: virtual _ _thiscall C……::～C……(void)"

连接时发现没有实现的外部符号（变量、函数等）。

参 考 文 献

[1] 谭浩强. C 程序设计[M]. 4 版. 北京：清华大学出版社，2010.

[2] 谭浩强. C 程序设计[M]. 3 版. 北京：清华大学出版社，2007.

[3] 孟祥双. C 语言教程[M]. 北京：北京师范大学出版社，2005.

[4] 汪新民，刘若慧. C 语言基础案例教程[M]. 北京：北京大学出版社，2011.

[5] 崔武子，齐华山. C 程序设计试题精选[M]. 清华：清华大学出版社，2009.

[6] Kenneth A. Reak. C 和指针[M]. 北京：人民邮电出版社，2003.

[7] 吕凤煮. C 语言基础教程[M]. 北京：北京大学出版社，1998.

[8] 霍顿. C 语言入门经典（第 4 版）. 北京：清华大学出版社，2008.

[9] 谭浩强. C 语言程序设计实践教程. 北京：机械工业出版社，2010.

[10] 恰汗·合孜尔. C 语言程序设计[M]. 北京：中国铁道出版社，2010.

[11] 马靖善，秦玉平，等. C 语言程序设计[M]. 北京：清华大学出版社，2005.

[12] 杜友福. C 语言程序设计. 2 版. 北京：科学出版社，2007.

[13] 谭浩强. C 语言程序设计. 2 版. 北京：清华大学出版社，2008.

[14] 王娣. C 语言从入门到精通. 北京：清华大学出版社，2010.

[15] 明日科技. C 语言从入门到精通. 北京：清华大学出版社，2012.

[16] 郭强. C 语言趣味编程 100 例. 北京：清华大学出版社，2014.

[17] 刘蕾. 21 天学通 C 语言（第 3 版）. 北京：电子工业出版社，2014.

[18] K. N. King. C 语言程序设计：现代方法（第 2 版）. 北京：人民邮电出版社，2010.

[19] 康莉，李宽. 零基础学 C 语言. 北京：机械工业出版社，2012.

[20] 康莉. 零基础学 C 语言（第 3 版）. 北京：机械工业出版社，2014.